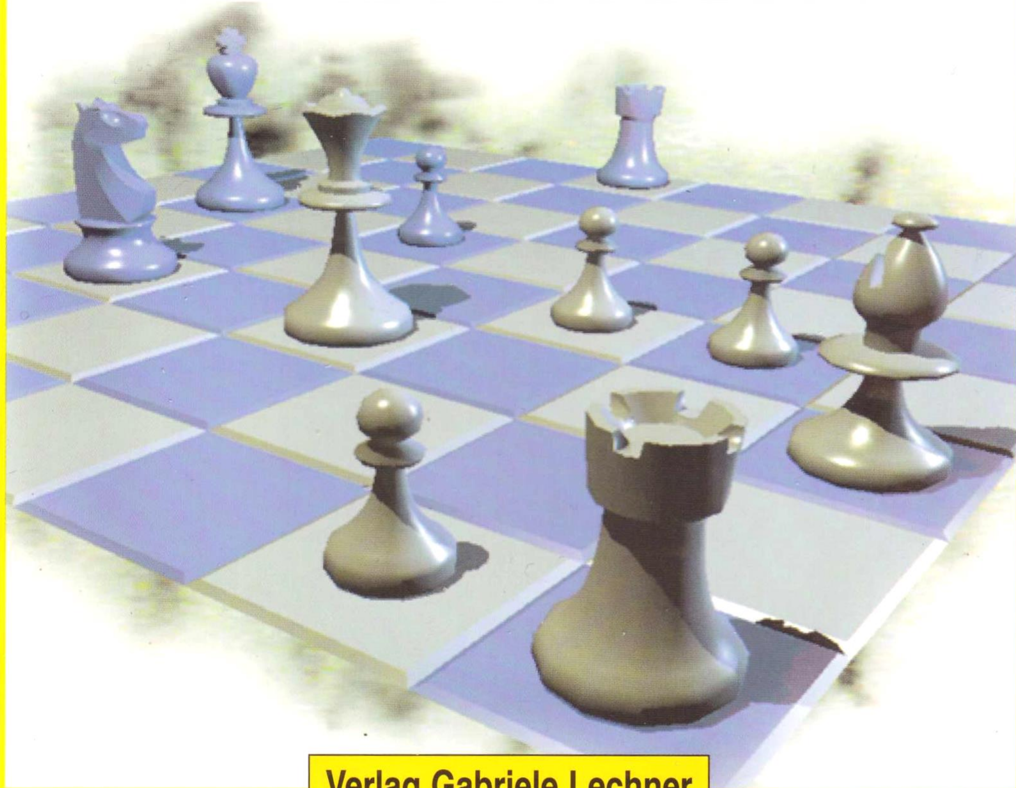
Niki Laber

AMIGA SPIELE SELBER PROGRAMMIEREN

Band 2

Schritt für Schritt von Null auf Hundert



Verlag Gabriele Lechner

SPIELE SELBER PROGRAMMIEREN

BAND 2

Niki Laber

Verlag Gabriele Lechner

*

ISBN 3-926858-64-8
1. Auflage 1996

Autor: Niki Laber
Illustrationen: Niki Laber
Einbandgestaltung: Björn Kindler
Druck: Druckerei Rampf, Dachau

Die Deutsche Bibliothek - CIP-Einheitsaufnahme
Amiga-Spiele selber programmieren / Niki Laber. - München :
Lechner.
NE: Laber, Niki
Bd. 2.
Buch. - 1. Aufl. - 1996

INHALTSVERZEICHNIS

VORWORT

Der Wunsch vieler Amigabesitzer ist es, auf dem Gerät, das für seine umwerfenden Grafikleistungen bekannt ist, selber Spiele zu entwickeln.

Mit der Buchreihe "Amiga Spiele selber programmieren" werden sowohl Fortgeschrittene als auch Programmiereinsteiger in die Programmierung eines Spieles eingeführt. In Band 1 wurden bereits die Grundlagen erläutert, weshalb darauf in diesem Buch nicht mehr näher eingegangen wird. Band 2 beschäftigt sich mit den Tricks und Feinheiten, die nötig sind, um den Amiga wirklich auszureizen.
Den Anfang bildet eine kurze Einführung in die Musikprogrammierung. hier lernen Sie, wie eine Abspielroutine für Module aufgebaut ist, damit Sie eigene Musikstücke in Ihre Programme integrieren können.

Ein großes Kapitel ist dem Blitter, einem der wichtigsten Bausteine des Amiga, gewidmet. Daß man mit ihm nicht nur Speicherbereiche verschieben, sondern auch Linien ziehen, Flächen füllen oder Objekte drehen kann, wird in diesem Kapitel ausführlich demonstriert.
Anhand von einigen praktischen Programmbeispielen wird das Zusammenspiel der einzelnen Chips wie Blitter und Copper erklärt. Alle Programmlistings, Grafiken und Musikstücke sowie einige nützliche Tools finden Sie auf der beigelegten Programmdiskette.

Großer Dank gebührt Hannes Seifert, der mit wertvoller Hilfe und fachmännischer Unterstützung wesentlich zu diesem Buch beigetragen hat.

Ich wünsche Ihnen viel Spaß und gutes Gelingen mit diesem Buch.

Wien, im April 1996

1. DER MAXON ASSEMBLER

WARUM ÜBERHAUPT ASSEMBLER?

Assembler ist die Sprache, mit der man einen Computer am maschinennächsten programmieren kann. Besonders der Amiga mit seinem komfortablen 68000er Prozessor eignet sich hervorragend für die Assemblerprogrammierung.
Gerade durch diese Programmiersprache kann man sich ungehindert auf dem Gerät bewegen und es sind fast alle Tricks möglich. Besonders bei der Spieleprogrammierung ist dies hilfreich. Hier ist es nicht nur wichtig, auf besondere technische Tricks zurückgreifen zu können, sondern daß man auch extrem kurzen Code schreiben kann.

ASSEMBLER, ABER WELCHER?

Die Vielzahl der am Markt befindlichen Assembler macht einem Anfänger die Entscheidung für das richtige Programm nicht gerade einfach. Sämtliche in diesem Buch abgebildeten Listings sind für den Seka, bzw. ASMOne Assembler geschrieben. Vor allem der Seka-Assembler ist schon sehr alt und wahrscheinlich heute überhaupt nicht mehr zu beziehen. Dennoch handelt es sich hierbei um ein Programm, das wahrscheinlich das am weitesten verbreitetste Programmiertool ist.

Es sollte aber keinerlei Schwierigkeiten bereiten, die Programme auch unter einem anderen Assembler zum Laufen zu bringen. Wichtigster Grundsatz hierbei ist, daß das fertige Programm aus dem Chip-Memory gestartet wird.

DER MAXON ASSEMBLER

Der von der Firma Maxon erhältliche Assembler ist ein komplexes Entwicklungspaket, dem ein gut durchdachter Modulaufbau zu Grunde liegt. Das Programm besteht aus drei großen Teilen:

- Der Editor
- Der Assembler
- Der Monitor

Alle drei Teile wurden in einem einzigen Programm realisiert, können aber bei Bedarf abgeschaltet werden, um Speicherplatz zu sparen.

DER EDITOR

Der mitgelieferte Editor ist angenehm über Maus oder Tastatur-shortcuts zu bedienen und verfügt über eine angenehme Verarbeitungsgeschwindigkeit, was gerade bei der Assemblerprogrammierung sehr wichtig ist.

Die umfangreichen Zusatzfunktionen, wie z.B. Makros, wurden in verschiedene Menüs unterteilt. Es gibt auch eine Fülle an praktischen Funktionen, die man erst im Laufe der fortgeschrittenen Programmierung zu schätzen lernt, z.B. die Funktion DC.x erzeugen.

DER ASSEMBLER

Dies ist wohl der wichtigste Teil des Entwicklungspaketes. Durch einen Menüpunkt kann man zwischen dem Assembler und dem Editor umschalten.

Das Assemblieren der Quelldatei selbst geht recht flott. Wurde ein Fehler entdeckt, wird dieser vom Programm angezeigt und man kann gleich in den Editor zurückschalten, um den Fehler zu korrigieren.

Eine sehr angenehme Funktion ist der Copperassembler. Mit diesem Programmteil erhält man eine wichtige Hilfe bei der Programmierung von Copperlisten. Man kann eine Copperliste anzeigen lassen und mit diesem Tool verändern. Fortgeschrittene Programmierer werden das zu schätzen wissen.

Ebenfalls sehr lobenswert ist die Optimierungsfunktion. Ist diese aktiviert, erkennt das Programm automatisch „schlecht" oder unpassend verwendete Befehle und ersetzt diese durch einen optimalen Befehl.

DER MONITOR

Der dritte und letzte große Programmteil ist der Monitor. Dieser unterstützt nicht nur die ohnehin selbstverständlichen Funktionen wie Speicherdarstellung in Hex oder ASCII, sowie Diskettenoperationen wie Block lesen oder schreiben, sondern dieses Modul verfügt auch über einen sehr umfangreichen Debugger. Mit diesem ist man in der Lage, sein eigenes Programm nach möglichen Fehlern zu durchforsten. Wenn ein Programm einmal zu einer unübersichtlichen Zahlen- und Buchstabenkolone angewachsen ist, kann man einen Fehler nur noch mit Hilfe eines Debuggers lokalisieren und beseitigen. Hier werden auch alle gängigen Funktionen unterstützt, wie zum Beispiel Einzelschrittabarbeitung oder das Setzen von Breakpoints.

Ein weiteres interessantes Feature ist der Reassembler. Mit diesem Tool können Sie bereits fertig erstellte Programm wieder zurück in Quellcode verwandeln und bei Bedarf verändern. Dieses Werkzeug ist aber eher für den fortgeschrittenen Anwender interessant, da es

sehr genaue Kenntnisse der Programmierung erfordert, einen fertigen Code zu interpretieren.

DAS HANDBUCH

Das mitgelieferte Handbuch ist zwar nicht allzu umfangreich, erklärt aber sämtliche Programmteile ausreichend in leicht verständlicher Sprache. Durch die vielen Abbildungen sollte es auch einem Assembler-Einsteiger leicht fallen, sich mit der Materie auseinander zu setzen.

Der gut gestaltete Anhang des Handbuches erklärt nicht nur den Befehlssatz des 68000er und 68010er, sondern zeigt auch sämtliche Fehlermeldungen auf, die das Programm ausgeben kann. Außerdem wurden Updateversionen des Assemblers in den zwei letzten Kapiteln berücksichtigt und die neuen Features erklärt.

Abschließend kann man sagen, daß der Maxon Assembler ein gutes Programmierwerkzeug ist, daß man Einsteigern und Profis gleichermaßen ans Herz legen kann.

2. MUSIKPROGRAMMIERUNG

Wie man hardwaremäßig Samples abspielt, haben wir schon in Band 1 besprochen. Nun ist es an der Zeit, auch das Abspielen von ganzen Musikstücken zu erläutern.

WAS IST MUSIKPROGRAMMIERUNG?

Das Abspielen eines Musikstückes unterscheidet sich nicht wesentlich vom Abspielen eines einzelnen Samples. Wie wir schon gehört haben, verfügt der Amiga über vier verschiedene Musikkanäle, die gleichzeitig Samples abspielen können.
Ein Stück ist aus mehreren Samples zusammengesetzt, die in den unterschiedlichsten Raten abgespielt werden. Dadurch ergibt sich, gute Komposition vorausgesetzt, ein wohlklingendes Lied.
Vorteil gegenüber einem einzigen langen Sample ist, daß man hierbei viel weniger Speicher verbraucht, da die einzelnen Samples beliebig untereinander kombiniert werden können.

Jedes Sample wird in eine Liste, auch Pattern genannt, eingetragen, wobei vermerkt wird, in welcher Tonhöhe es abgespielt werden soll. Für jeden der vier Kanäle werden die Samples bestimmt.

Komponiert werden die Musikstücke mit einem Soundtracker oder auch Modul-Player. Durch einfachste Bedienung brauchen die Musiker keine Ahnung von der Programmierung haben.

WIE IST EIN PATTERN AUFGEBAUT?

Anfänglich werden die Samples definiert. Jedes Sample erhält eine Nummer, unter der es später aufgerufen wird.

Zuerst steht die Note, die für die erste Stimme (Kanal 1) angeschlagen werden soll. Allerdings wurde der Wert (1 Word)

nicht als Note, sondern direkt als Rate eingetragen. Das nächste Byte enthält im Highnibble die Nummer jenes Samples, das angeschlagen werden soll. Das Lownibble zeigt an, welcher Effekt gespielt werden soll. Ein Effekt ist beispielsweise die Lautstärkenänderung. Das nächste Byte gibt an, welchen Wert der gewählte Effekt haben soll. Beim Beispiel der Lautstärke wäre das, um wieviel diese erhöht oder erniedrigt werden soll. Prinzipiell kann ein Lied aus beliebig viele Patterns bestehen.



Hier nun das vollständige Listing zum Abspielen eines fertigen Soundtracker Moduls:

```
;
;           Abspielen eines Musikmoduls
;
; Das File MUSIC muß nach mt_data nachgeladen werden!
;
;
s:          bsr.s       mt_init
;
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$00003000,d0
            bne.s       loop
;
            bsr.L       mt_music
;
```

```
        btst    #6,$bfe001
        bne.s   loop
;
end:    bsr.L   mt_end
;
e:      rts
;
mt_init:        lea     mt_data,a0
        lea     $3b8(a0),a1
        moveq   #$7f,d0
        moveq   #0,d2
        moveq   #0,d1
mt_lop2:        move.b  (a1)+,d1
        cmp.b   d2,d1
        ble.s   mt_lop
        move.l  d1,d2
mt_lop: dbf     d0,mt_lop2
        addq.b  #1,d2

        asl.l   #8,d2
        asl.l   #2,d2
        lea     4(a1,d2.l),a2
        lea     mt_samplestarts(pc),a1
        add.w   #42,a0
        moveq   #$1e,d0
mt_lop3:        clr.l   (a2)
        move.l  a2,(a1)+
        moveq.  #0,d1
        move.w  (a0),d1
        clr.b   2(a0)
        asl.l   #1,d1
        add.l   d1,a2
        add.l   #$1e,a0
        dbf     d0,mt_lop3
```

```
            or.b      #2,$bfe001
            move.b    #6,mt_speed
            moveq     #0,d0
            lea       $dff000,a0
            move.w    d0,$a8(a0)
            move.w    d0,$b8(a0)
            move.w    d0,$c8(a0)
            move.w    d0,$d8(a0)
            clr.b     mt_songpos
            clr.b     mt_counter
            clr.w     mt_pattpos
            rts

mt_end:     clr.w     $dff0a8
            clr.w     $dff0b8
            clr.w     $dff0c8
            clr.w     $dff0d8
            move.w    #$f,$dff096
            rts

mt_music:
            lea       mt_data,a0
            addq.b    #1,mt_counter
            move.b    mt_counter(pc),d0
            cmp.b     mt_speed(pc),d0
            blt       mt_nonew
            clr.b     mt_counter

            lea       mt_data,a0
            lea       $c(a0),a3
            lea       $3b8(a0),a2
            lea       $43c(a0),a0

            moveq     #0,d0
            moveq     #0,d1
            move.b    mt_songpos(pc),d0
```

```
        move.b  (a2,d0.w),d1
        lsl.w   #8,d1
        lsl.w   #2,d1
        add.w   mt_pattpos(pc),d1
        clr.w   mt_dmacon

        lea     $dff0a0,a5
        lea     mt_voice1(pc),a4
        bsr     mt_playvoice
        addq.l  #4,d1
        lea     $dff0b0,a5
        lea     mt_voice2(pc),a4
        bsr     mt_playvoice
        addq.l  #4,d1
        lea     $dff0c0,a5
        lea     mt_voice3(pc),a4
        bsr     mt_playvoice
        addq.l  #4,d1
        lea     $dff0d0,a5
        lea     mt_voice4(pc),a4
        bsr     mt_playvoice

        bsr     mt_wait
        move.w  mt_dmacon(pc),d0
        or.w    #$8000,d0
        move.w  d0,$dff096
        bsr     mt_wait
mt_nodma:
        lea     mt_voice1(pc),a4
        lea     $dff000,a3
        move.l  $a(a4),$a0(a3)
        move.w  $e(a4),$a4(a3)
        move.w  $12(a4),$a8(a3)
        move.l  $a+$1c(a4),$b0(a3)
        move.w  $e+$1c(a4),$b4(a3)
        move.w  $12+$1c(a4),$b8(a3)
```

```
            move.l  $a+$38(a4),$c0(a3)
            move.w  $e+$38(a4),$c4(a3)
            move.w  $12+$38(a4),$c8(a3)
            move.l  $a+$54(a4),$d0(a3)
            move.w  $e+$54(a4),$d4(a3)
            move.w  $12+$54(a4),$d8(a3)

            add.w   #$10,mt_pattpos
            cmp.w   #$400,mt_pattpos
            bne.s   mt_exit
mt_next:    clr.w   mt_pattpos
            clr.b   mt_break
            lea     mt_data,a0
            addq.b  #1,mt_songpos
            and.b   #$7f,mt_songpos
            move.b  $3b6(a0),d0
            cmp.b   mt_songpos(pc),d0
            bne.s   mt_exit
            move.b  $3b7(a0),mt_songpos
mt_exit:    tst.b   mt_break
            bne.s   mt_next
            rts

mt_wait:    moveq   #4,d3
mt_wai2:    move.b  $dff006,d2
mt_wai3:    cmp.b   $dff006,d2
            beq.s   mt_wai3
            dbf     d3,mt_wai2
            moveq   #8,d2
mt_wai4:    dbf     d2,mt_wai4

mt_nonew:
            lea     mt_voice1(pc),a4
            lea     $dff0a0,a5
            bsr     mt_com
            lea     mt_voice2(pc),a4
```

```
        lea     $dff0b0,a5
        bsr     mt_com
        lea     mt_voice3(pc),a4
        lea     $dff0c0,a5
        bsr     mt_com
        lea     mt_voice4(pc),a4
        lea     $dff0d0,a5
        bsr     mt_com
        bra.s   mt_exit

mt_mulu:
        dc.w
        0,$1e,$3c,$5a,$78,$96,$b4,$d2,$f0,$10e,$12c,$14a
        dc.w
        $168,$186,$1a4,$1c2,$1e0,$1fe,$21c,$23a,$258,$276
        dc.w
        $294,$2b2,$2d0,$2ee,$30c,$32a,$348,$366,$384,$3a2

mt_playvoice:
        move.l  (a0,d1.l),(a4)
        moveq   #0,d2
        move.b  2(a4),d2
        lsr.b   #4,d2
        move.b  (a4),d0
        and.b   #$f0,d0
        or.b    d0,d2
        beq.s   mt_oldinstr

        lea     mt_samplestarts-4(pc),a1
        asl.w   #2,d2
        move.l  (a1,d2.l),4(a4)
        lsr.w   #1,d2
        move.w  mt_mulu(pc,d2.w),d2
        move.w  (a3,d2.w),8(a4)
        move.w  2(a3,d2.w),$12(a4)
        moveq   #0,d3
```

```
		move.w	4(a3,d2.w),d3
		tst.w	d3
		beq.s	mt_noloop
		move.l	4(a4),d0
		asl.w	#1,d3
		add.l	d3,d0
		move.l	d0,$a(a4)
		move.w	4(a3,d2.w),d0
		add.w	6(a3,d2.w),d0
		move.w	d0,8(a4)
		bra.s	mt_hejaSverige
mt_noloop:
		move.l	4(a4),d0
		add.l	d3,d0
		move.l	d0,$a(a4)
mt_hejaSverige:
		move.w	6(a3,d2.w),$e(a4)

mt_oldinstr:
		move.w	(a4),d0
		and.w	#$fff,d0
		beq	mt_com2
		tst.w	8(a4)
		beq.s	mt_stopsound
		tst.b	$12(a4)
		bne.s	mt_stopsound
		move.b	2(a4),d0
		and.b	#$f,d0
		cmp.b	#5,d0
		beq.s	mt_setport
		cmp.b	#3,d0
		beq.s	mt_setport

		move.w	(a4),$10(a4)
		and.w	#$fff,$10(a4)
		move.w	$1a(a4),$dff096
```

```
        clr.b     $19(a4)
        move.l    4(a4),(a5)
        move.w    8(a4),4(a5)
        move.w    $10(a4),6(a5)

        move.w    $1a(a4),d0
        or.w      d0,mt_dmacon
        bra       mt_com2

mt_stopsound:
        move.w    $1a(a4),$dff096
        bra       mt_com2

mt_setport:
        move.w    (a4),d2
        and.w     #$fff,d2
        move.w    d2,$16(a4)
        move.w    $10(a4),d0
        clr.b     $14(a4)
        cmp.w     d0,d2
        beq.s     mt_clrport
        bge       mt_com2
        move.b    #1,$14(a4)
        bra       mt_com2
mt_clrport:
        clr.w     $16(a4)
        rts

mt_port: move.b   3(a4),d0
        beq.s     mt_port2
        move.b    d0,$15(a4)
        clr.b     3(a4)
mt_port2:
        tst.w     $16(a4)
        beq.s     mt_rts
        moveq     #0,d0
```

```
            move.b  $15(a4),d0
            tst.b   $14(a4)
            bne.s   mt_sub
            add.w   d0,$10(a4)
            move.w  $16(a4),d0
            cmp.w   $10(a4),d0
            bgt.s   mt_portok
            move.w  $16(a4),$10(a4)
            clr.w   $16(a4)
mt_portok:
            move.w  $10(a4),6(a5)
mt_rts:     rts

mt_sub:     sub.w   d0,$10(a4)
            move.w  $16(a4),d0
            cmp.w   $10(a4),d0
            blt.s   mt_portok
            move.w  $16(a4),$10(a4)
            clr.w   $16(a4)
            move.w  $10(a4),6(a5)
            rts

mt_sin:
dc.b        0,$18,$31,$4a,$61,$78,$8d,$a1,$b4,$c5,$d4,$e0,$eb,$f4
dc.b        $fa,$fd
dc.b        $ff,$fd,$fa,$f4,$eb,$e0,$d4,$c5,$b4,$a1,$8d,$78,$61
dc.b        $4a,$31,$18

mt_vib:     move.b  $3(a4),d0
            beq.s   mt_vib2
            move.b  d0,$18(a4)

mt_vib2:    move.b  $19(a4),d0
            lsr.w   #2,d0
            and.w   #$1f,d0
            moveq   #0,d2
```

```
        move.b  mt_sin(pc,d0.w),d2
        move.b  $18(a4),d0
        and.w   #$f,d0
        mulu    d0,d2
        lsr.w   #7,d2
        move.w  $10(a4),d0
        tst.b   $19(a4)
        bmi.s   mt_vibsub
        add.w   d2,d0
        bra.s   mt_vib3
mt_vibsub:
        sub.w   d2,d0
mt_vib3: move.w d0,6(a5)
        move.b  $18(a4),d0
        lsr.w   #2,d0
        and.w   #$3c,d0
        add.b   d0,$19(a4)
        rts

mt_arplist:
        dc.b    0,1,2,0,1,2,0,1,2,0,1,2,0
        dc.b    1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1

mt_arp: moveq   #0,d0
        move.b  mt_counter(pc),d0
        move.b  mt_arplist(pc,d0.w),d0
        beq.s   mt_arp0
        cmp.b   #2,d0
        beq.s   mt_arp2
mt_arp1: moveq  #0,d0
        move.b  3(a4),d0
        lsr.b   #4,d0
        bra.s   mt_arpdo
mt_arp2: moveq  #0,d0
        move.b  3(a4),d0
        and.b   #$f,d0
```

```
mt_arpdo:
            asl.w       #1,d0
            move.w      $10(a4),d1
            and.w       #$fff,d1
            lea         mt_periods(pc),a0
            moveq       #$24,d2
mt_arp3:    cmp.w       (a0)+,d1
            bge.s       mt_arpfound
            dbf         d2,mt_arp3
mt_arp0:    move.w      $10(a4),6(a5)
            rts
mt_arpfound:
            move.w      -2(a0,d0.w),6(a5)
            rts

mt_normper:
            move.w      $10(a4),6(a5)
            rts

mt_com:     move.w      2(a4),d0
            and.w       #$fff,d0
            beq.s       mt_normper
            move.b      2(a4),d0
            and.b       #$f,d0
            tst.b       d0
            beq.s       mt_arp
            cmp.b       #1,d0
            beq.s       mt_portup
            cmp.b       #2,d0
            beq.s       mt_portdown
            cmp.b       #3,d0
            beq         mt_port
            cmp.b       #4,d0
            beq         mt_vib
            cmp.b       #5,d0
            beq.s       mt_volport
```

```
        cmp.b   #6,d0
        beq.s   mt_volvib
        move.w  $10(a4),6(a5)
        cmp.b   #$a,d0
        beq.s   mt_volslide
        rts

mt_portup:
        moveq   #0,d0
        move.b  3(a4),d0
        sub.w   d0,$10(a4)
        move.w  $10(a4),d0
        cmp.w   #$71,d0
        bpl.s   mt_portup2
        move.w  #$71,$10(a4)
mt_portup2:
        move.w  $10(a4),6(a5)
        rts

mt_portdown:
        moveq   #0,d0
        move.b  3(a4),d0
        add.w   d0,$10(a4)
        move.w  $10(a4),d0
        cmp.w   #$358,d0
        bmi.s   mt_portdown2
        move.w  #$358,$10(a4)
mt_portdown2:
        move.w  $10(a4),6(a5)
        rts

mt_volvib:
        bsr     mt_vib2
        bra.s   mt_volslide
mt_volport:
        bsr     mt_port2
```

```
mt_volslide:
        moveq   #0,d0
        move.b  3(a4),d0
        lsr.b   #4,d0
        beq.s   mt_vol3
        add.b   d0,$13(a4)
        cmp.b   #$40,$13(a4)
        bmi.s   mt_vol2
        move.b  #$40,$13(a4)
mt_vol2: moveq  #0,d0
        move.b  $13(a4),d0
        move.w  d0,8(a5)
        rts
mt_vol3: move.b 3(a4),d0
        and.b   #$f,d0
        sub.b   d0,$13(a4)
        bpl.s   mt_vol4
        clr.b   $13(a4)
mt_vol4: moveq  #0,d0
        move.b  $13(a4),d0
        move.w  d0,8(a5)
        rts

mt_com2: move.b $2(a4),d0
        and.b   #$f,d0
        cmp.b   #$e,d0
        beq.s   mt_filter
        cmp.b   #$d,d0
        beq.s   mt_pattbreak
        cmp.b   #$b,d0
        beq.s   mt_songjmp
        cmp.b   #$c,d0
        beq.s   mt_setvol
        cmp.b   #$f,d0
        beq.s   mt_setspeed
        rts
```

```
mt_filter:
        move.b  3(a4),d0
        and.b   #1,d0
        asl.b   #1,d0
        and.b   #$fd,$bfe001
        or.b    d0,$bfe001
        rts

mt_pattbreak:
        move.b  #1,mt_break
        move.w  #$400-$10,mt_pattpos
        rts

mt_songjmp:
        move.b  #1,mt_break
        move.b  3(a4),d0
        subq.b  #1,d0
        move.b  d0,mt_songpos
        rts

mt_setvol:
        cmp.b   #$40,3(a4)
        bls.s   mt_sv2
        move.b  #$40,3(a4)
mt_sv2: moveq   #0,d0
        move.b  3(a4),d0
        move.b  d0,$13(a4)
        move.w  d0,8(a5)
        rts

mt_setspeed:
        moveq   #0,d0
        move.b  3(a4),d0
        cmp.b   #$1f,d0
        bls.s   mt_sp2
        moveq   #$1f,d0
```

```
mt_sp2:     tst.w       d0
            bne.s       mt_sp3
            moveq       #1,d0
mt_sp3:     move.b      d0,mt_speed
            rts

mt_periods:
dc.w    $358,$328,$2fa,$2d0,$2a6,$280,$25c,$23a,$21a,$1fc,$1e0
dc.w    $1c5,$1ac,$194,$17d,$168,$153,$140,$12e,$11d,$10d,$fe
dc.w    $f0,$e2,$d6,$ca,$be,$b4,$aa,$a0,$97,$8f,$87
dc.w    $7f,$78,$71,0

mt_speed:           dc.b        6
mt_counter:         dc.b        0
mt_pattpos:         dc.w        0
mt_songpos:         dc.b        0
mt_break:           dc.b        0
mt_dmacon:          dc.w        0
mt_samplestarts:    blk.l       $1f,0
mt_voice1:          blk.w       13,0
                    dc.w        1
mt_voice2:          blk.w       13,0
                    dc.w        2
mt_voice3:          blk.w       13,0
                    dc.w        4
mt_voice4:          blk.w       13,0
                    dc.w        8
mt_data:            blk.b       59526+[2*1024],0
```

3. DER BLITTER

3.1. DIE BLITTER-REGISTER

Um den Blitter anzusprechen, muß man ihm zuerst einige Parameter übergeben. Dazu sind einige Register bestimmt, die die Informationen aufnehmen.

BLTSIZE ($dff058)

Das erste Register ist das Blitter-Size Register, welches dem Blitter die Größe des zu kopierenden Bereiches übermittelt. Aber es dient nicht nur zum Markieren des Bereiches, sondern auch zum Starten des Blitters, daher muß dieses Register als letztes initialisiert werden, da dem Blitter sonst noch nötige Informationen fehlen.
Wie sieht aber das Blitterfenster aus? Das Blitterfenster ist jener Speicherbereich, der vom Blitter während der Operation verwendet wird. Eingeteilt wird der Speicher, genau wie bei den Bitplanes, in Spalten und Zeilen. Eine Spalte entspricht einem Word = zwei Byte.
Sehen wir uns nun die genauere Biteinteilung des BLTSIZE-Registers an:

```
Bits:
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
H9 H8 H7 H6 H5 H4 H3 H2 H1 H0 W5 W4 W3 W2 W1 W0
```

Die ersten zehn Bits bestimmen die Höhe des Blitterfensters. Die Höhe wird in Zeilen angegeben, welche zwischen 0 und 2^{10} (1024) liegen kann. Will man 1024 Zeilen anwählen, so sind alle Bits auf Null zu setzten. Eine Höhe von Null Zeilen ist daher nicht mehr möglich.

Die restlichen sechs Bits stehen für die Breite, welche ebenfalls eine Größe von 1 bis 1024 Pixel haben darf. Die Breite wird in Words angegeben; 1 Word hat 16 Bits (16 Pixel) - maximal 64 Words ergeben: 64*16=1024. Es gilt hier die gleiche Regelung wie bei der

Höhe: Will man 64 Words wählen, so sind diese Bits auf Null zu setzten. Eine Breite von Null Words ist wie oben unmöglich.

Damit man nicht bei jeder Blitteroperation umständlich die Bitbelegung studieren muß, gibt es eine einfache Formel, mit der man den richtigen Wert für BLTSIZE errechnen kann:

BLTSIZE = Höhe*64 + Breite in Words

Beispiel: Der Blitbereich ist 320 Pixel (320/16 ergibt die Breite in Words) breit und 10 Pixel hoch, dann ergibt das:

BLTSIZE = 10*64 + 20
BLTSIZE = 660

BLTAPTH ($dff050)

Dieses Register enthält zusammen mit dem Register BLTAPTL ($dff052) die Startadresse der Quelle A. Es existieren insgesamt vier Register (A, B, C und D). Davon sind drei (A, B und C) die Quellregister und eines (D) das Zielregister. Der Blitter holt die Informationen aus den entsprechenden Quellregistern, verknüpft sie wahlweise miteinander und schreibt das Ergebnis in das Zielregister D. In BLTAPTH liegen die high Bits der Startadresse. Am Ende einer Blitteroperation liegt in diesen Registern die Adresse des letzten Words plus 2, plus dem entsprechenden Modulo-Wert (die Erklärungen zu den Modulo-Werten folgt etwas weiter unten).

BLTAPTL ($dff052)

In diesem Register liegen die low Bits der Startadresse von Quellregister A. Da diese Register hintereinander liegen, ist es nicht nötig beide getrennt zu beschreiben, sondern man kann dies mit einem einzigen Befehl erledigen:

```
move.l #pic,$dff050
```

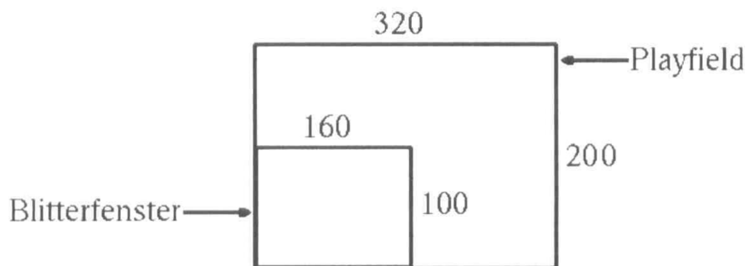
BLTBPTH ($dff04c)
BLTBPTL ($dff04e)
BLTCPTH ($dff048)
BLTCPTL ($dff04a)
BLTDPTH ($dff054)
BLTDPTL ($dff056)

Die oben angeführten Register sind genauso zu behandeln wie BLTAPTH und BLTAPTL.

BLTAMOD ($dff064)

In diesem Register befindet sich der Modulo-Wert für die Quelle A. Da der Blitter, genau wie die Playfields, auch mit rechteckigen Speicherbereichen arbeitet, müssen wir ihm einen Modulowert zuteilen.

Doch wie arbeitet man mit einem Modulo Wert? Ganz einfach, durch die rechteckige Speichereinteilung ist es möglich, innerhalb eines größeren Bereiches einen kleineren zu definieren, der sowohl eigene Breite als auch eigene Höhe besitzt. Sehen wir uns das Ganze anhand eines Beispiels an:



Nehmen wir an, unser Playfield besitzt eine Größe von 320 mal 200 Punkten. In dieses wollen wir einen kleineren Ausschnitt mit der Größe von 160 mal 100 Punkten hineinkopieren. Der Blitter kopiert

dieses kleinere Fenster zeilenweise in den Zielbereich. Damit er aber jede Zeile richtig untereinandersetzt, gibt ihm der Modulowert an, wieviel er nach jeder kopierten Zeile hinzuzählen muß, damit er die nächste Zeile wieder an die richtige Stelle kopiert. Der Modulowert ist demnach nichts anderes, als die Differenz von Ziel- und Blitterfenster. Da der Modulowert in Bytes angegeben wird, müssen wir das Ergebnis noch durch 8 dividieren. Die korrekte Formel zur Moduloberechnung lautet daher:

(ZielfensterX - BlitterfensterX)/8 = Modulo

Für unser obiges Beispiel werden folgende Zahlen eingesetzt: (320 - 160)/8 = Modulo
20 = Modulo

BLTBMOD ($dff062)
BLTCMOD ($dff060)
BLTDMOD ($dff066)

Diese Modulo-Werte kann man selbstverständlich für alle Quellen und das Zielregister seperat einstellen, daher existieren auch vier getrennte Register.

BLTAFWM ($dff044)

Da man mit dem Blitter, wie wir schon weiter oben erfahren haben, nur wortweise kopieren kann, können die kopierten Daten nur ein Vielfaches von 16 Punkten breit sein. Wenn wir nun eine beliebige Breite verschieben möchten, dann müssen wir die an den Rändern "überhängenden" Pixel weglassen (in der Fachsprache auch ausmaskieren genannt). Mit dem Register BLTAFWM wird das erste Word ausmaskiert. Die Maskierung funktioniert ziemlich einfach: Jedes auf 1 gesetzte Bit wird beim Kopiervorgang übernommen. Alle nicht gesetzten Bits werden gelöscht. Ein Beispiel verschafft Klarheit:

Nehmen wir an, unser Objekt besteht aus drei Words, wobei sowohl das erste als auch das letzte nur zur Hälfte übernommen wird, das mittlere hingegen ganz kopiert werden soll.

Vor der Operation:

Word 1	Word 2	Word 3
0000111001110000	0000111001110000	0000111001110000
0001111111111000	0001111111111000	0001111111111000
0011111001111100	0011111001111100	0011111001111100
0111110000111110	0111110000111110	0111110000111110
1111100000011111	1111100000011111	1111100000011111
0111110000111110	0111110000111110	0111110000111110
0011111001111100	0011111001111100	0011111001111100
0001111111111000	0001111111111000	0001111111111000
0000111001110000	0000111001110000	0000111001110000

Nach der Operation durch Verwendung folgender Masks:

BLTAFWM:		BLTALWM:
0000000011111111		1111111100000000

0000000001110000	0000111001110000	0000111000000000
0000000011111000	0001111111111000	0001111100000000
0000000001111100	0011111001111100	0011111000000000
0000000000111110	0111110000111110	0111110000000000
0000000000011111	1111100000011111	1111100000000000
0000000000111110	0111110000111110	0111110000000000
0000000001111100	0011111001111100	0011111000000000
0000000011111000	0001111111111000	0001111100000000
0000000001110000	0000111001110000	0000111000000000

BLTALWM ($dff046)

Dieses Register bestimmt die Maske des letzten Words. Sowohl BLTAFWM als auch BLTALWM sollten im Line- und im Fill-Modus auf eins gesetzt werden.

BLTCON0 ($dff040)

Wie wir eben erfahren haben, kann man mit dem Blitter auch Grafiken kopieren, die nicht unbedingt an einer Wortgrenze (also einem Vielfachen von 16 Punkten) enden. Aber als Zielbereich können wir wiederum nur eine an einer Wortgrenze liegende Adresse wählen. Das bedeutet, daß wir nur alle 16 Pixel etwas hineinblitten können. Was ist aber nun zu tun, wenn man auch dazwischen Daten ablegen möchte?

Auch dafür stellt uns der Blitter eine Funktion zur Verfügung. Mit den Bits 12-15 des BLTCON0 Registers wird bestimmt, um wieviele Bits die Daten verschoben werden sollen. Die hinausgeschobenen Bits werden beim nächsten Word wieder hineingeshiftet. Sehen wir uns ein Beispiel an:

Vor dem Verschieben:	0001010111000000
Nach dem Verschieben:	xxxx000101011100

In unserem Beispiel wurde das Wort um vier Bits verschoben. Die "xxxx" repräsentieren jene Bits, die im vorhergehenden Wort stehen. Eine Verschiebung gibt es nur für Quelle A und B. Die Bits zur Steuerung von Quelle B befinden sich im Register BLTCON1.

Wie schon oben erwähnt, bietet uns der Blitter die Möglichkeit, die drei Quellregister logisch miteinander zu verknüpfen. Das Ergebnis dieser Verknüpfung wird in dem Zielregister D abgelegt. Uns stehen acht unterschiedliche Bit-Tripel zur Verfügung ($2^3 = 8$). Diese sind in den sogenannten LFx-Bits (Bits 0-7) enthalten. Diese Bit-Tripel

nennt man Miniterms. Mittels dieser Miniterms kann der Anwender bestimmen, wie das Ergebnis in D gebildet werden soll (Siehe dazu Kapitel 3.3. Die Mini-Terms).

Sehen wir uns die Bit-Belegung des Registers in gesamter Form an:

Bit	Name	Funktion
15	ASH3	Diese vier Bits beinhalten den Wert der Verschiebung von Quelle A. Sind alle 4 Bits auf Null gesetzt, so findet keine Verschiebung statt.
14	ASH2	
13	ASH1	
12	ASH0	
11	USEA	DMA-Kanal für Quelle A einschalten.
10	USEB	DMA-Kanal für Quelle B einschalten.
9	USEC	DMA-Kanal für Quelle C einschalten.
8	USED	DMA-Kanal für Ziel D einschalten.
7	LF7	Miniterm ABC (=111)
6	LF6	Miniterm ABc (=110)
5	LF5	Miniterm AbC (=101)
4	LF4	Miniterm Abc (=100)
3	LF3	Miniterm aBC (=011)
2	LF2	Miniterm aBc (=010)
1	LF1	Miniterm abC (=001)
0	LF0	Miniterm abc (=000)

BLTCON1 ($dff042)

Bit	Name	Funktion
15	BSH3	Diese vier Bits beinhalten den Wert der Verschiebung von Quelle B. Sind alle 4 Bits auf Null gesetzt, so findet keine Verschiebung statt.
14	BSH2	
13	BSH1	
12	BSH0	
11 - 5		unbenützt
4	EFE	Exclusive Fill Enable Die Bits 2, 3 und 4 sind nur in Zusammenhang mit der Funktion "Flächen füllen" interessant. Wollen Sie aber normal arbeiten, so sind beide Bits auf Null zu setzen. (Flächen füllen siehe Kapitel 5.3.)
3	IFE	Inclusive Fill Enable
2	FCI	Fill Carry In
1	DESC	Wenn dieses Bit eins ist, dann ist der descending Modus aktiviert. Wenn nicht, arbeitet der Blitter normal im ascending Modus (Erklärung unten).
0	LINE	Wird dieses Bit auf eins gesetzt, so wird der Line-Modus aktiviert und der Blitter kann zum Ziehen von Linien verwendet werden (siehe auch Kapitel 5.3. Line-Befehl).

ASCENDING-, DESCENDING-MODUS

Was bedeuten nun diese beiden Modi? Normalerweise arbeitet der Blitter aufsteigend (ascending), daß heißt, er beginnt mit dem Kopieren der Daten bei der Anfangsadresse und erhöht diese solange, bis er bei der Endadresse angelangt ist. Will man aber einen Speicherbereich kopieren, bei dem sich Quelle und Ziel teilweise überlappen, so führt dies nicht zum gewünschten Ergebnis. Sehen wir uns dazu die Grafik an:

Ausgangssituation			
Quelle A			
Quelle B			
Quelle C			Ergebnis
Quelle D	Ziel A	Quelle A	Quelle A
Quelle E	Ziel B	Quelle B	Quelle B
	Ziel C	Quelle C	Quelle C
	Ziel D	Quelle D	Quelle A
	Ziel E	Quelle E	Quelle B

Im obigen Beispiel wird versucht, eine Graphik in einen Bereich zu kopieren, der sich aber mit den Quell-Daten überlappt. Daher wird bei verwendetem Ascending-Modus anstatt der kompletten Quelldaten ein Teil schon überschrieben, bevor er kopiert werden kann. Daher steht am Ende der Zieldaten der Teil, der zuerst kopiert wurde.

Deshalb kann man den Blitter auch umgekehrt arbeiten lassen, nämlich absteigend (descending). Er beginnt jetzt bei der Endadresse und subtrahiert solange bis er bei der Startadresse angelangt ist. Dadurch wird das Ergebnis, wie in der Zeichnung, korrekt übertragen. Damit der Blitter auch korrekt arbeitet, muß man ihm natürlich bei eingeschaltetem decending-Modus mit der Endadresse zu kopieren beginnen lassen, da er nun absteigend arbeitet.

BLTADAT ($dff074)
BLTBDAT ($dff072)
BLTCDAT ($dff070)
BLTDDAT ($dff000)

Die Daten, die kopiert werden sollen, werden vom Blitter mit Hilfe von vier DMA-Kanälen aus dem Speicher geholt, bearbeitet (verknüpft) und anschließend wieder zurückgeschrieben. Nach der

Verarbeitung steht das Ergebnis in BLTDDAT und wird danach ins Ram (Chip-Ram) geschrieben. Man kann jeden dieser Kanäle mit den USEx-Bits des Registers BLTCON0 einzeln ein- oder ausschalten. Will man einen Kanal sperren, so ist das entsprechende USEx-Bit auf Null zu setzen, sowie bei der Auswahl der Miniterms darauf zu achten, daß diese keinen Einfluß auf diesen Kanal nehmen.

DMACON ($dff096)

Die folgenden Bits aus dem Register DMACON beeinflussen ebenfalls den Blitter:

Bit	Name	Funktion
14	BBUSY	Blitter status - testet ob der Blitter gerade arbeitet oder nicht (nur lesen).
13	BZERO	Blitter zero status - Das Ergebnis jeder Blitteroperation war 0 (nur lesen).
10	BLTPRI	Blitter priority - Der Blitter hat Vorrang gegenüber dem Prozessor.
6	BLTEN	Blitter DMA enable - Blitter DMA einschalten.

3.2. DIE MINI-TERMS

Der Blitter besitzt, wie wir bereits wissen, drei Quellen, aber nur ein Ziel. Diese Tatsache führt zu folgenden Fragen:

1) Warum benötigt man drei Quellen, wenn der Blitter immer nur ein Ziel gleichzeitig ansprechen kann?

2) Wie kann man den Einfluß der drei verschiedenen Quellen auf das Ziel bestimmen?

Die erste Frage läßt sich wohl am besten anhand eines Beispieles erklären:

Quellfenster A	Quellfenster B	gewünschtes Ergebnis
0101010101010101	0000000110000000	0000000100000000
0101010101010101	0000001110000000	0000000101000000
0101010101010101	0000011111100000	0000010101000000
0101010101010101	0000111111110000	0000010101010000
0101010101010101	0001111111111000	0001010101010000
0101010101010101	0011111111111100	0001010101010100
0101010101010101	0111111111111110	0101010101010100
0101010101010101	1111111111111111	0101010101010101

Wir haben zwei Quellen (A und B), wobei als Ergebnis nur diejenigen Bits im Zielregister gesetzt werden sollen, die in beiden Quellregistern auf 1 gesetzt sind. Um aus den beiden Quellen A und B das abgebildete Ergebnis zu bekommen, ist folgende Minitermkombination notwendig: 11000000

Die Beantwortung der Frage, warum ausgerechnet diese Kombination nötig ist, geht einher mit der Beantwortung der zweiten oben gestellten Frage:

Ähnlich den logischen Verknüpfungen (AND, OR, EOR,...) funktionieren auch die Miniterms. Der Vorteil ist jedoch, daß die Kombinationen, die 1 ergeben sollen, frei wählbar sind. Die Minitermkombination 00111111 würde zum Beispiel das oben gezeigte Ergebnis invertiert darstellen.

Es gilt, für alle möglichen Kombinationen der drei Quellen auszuwählen, ob das Ergebnis 1 oder 0 sein soll. Wird das jeweilige Bit gesetzt, so ist das Ergebnis immer eine 1. Hier nun eine Tabelle, die die zu jedem Bit gehörige Eingangskombination darstellt. (Zur Verdeutlichung wurden die Quellen nochmals dargestellt. Ein

Großbuchstabe bedeutet, das das jeweilige Bit der Eingangskombination 1 ist, ein Kleinbuchstabe bedeutet 0):

	ABC		Bitbezeichnung	Nummer
Eingangskomb.:	111	ABC	LF7	7
	110	ABc	LF6	6
	101	AbC	LF5	5
	100	Abc	LF4	4
	011	aBC	LF3	3
	010	aBc	LF2	2
	001	abC	LF1	1
	000	abc	LF0	0

Für unser oben angeführtes Beispiel wählen wir jene Bitkombinationen aus, die sowohl bei A als auch bei B ein wahres Ergebnis liefert:

	ABC		Bitbezeichnung	Nummer
===>	111	ABC	LF7	7
===>	110	ABc	LF6	6
	101	AbC	LF5	5
	100	Abc	LF4	4
	011	aBC	LF3	3
	010	aBc	LF2	2
	001	abC	LF1	1
	000	abc	LF0	0

Nur die Bits 6 und 7 liefern ein korrektes Ergebnis, daher lautet die richtige Einstellung der Miniterms:

```
move.w #%11000000,$dff040
```

Abschließend noch einige nützliche Beispiele für Miniterm-kombinationen:

1) Kombination: 11110000
Eingeschaltete Quellen: A
Ergebnis: Das Quellfenster A wird unverändert im Zielfenster D abgelegt.

2) Kombination: 00001111
Eingeschaltete Quellen: A
Ergebnis: Das Quellfenster A wird invertiert im Zielfenster D abgelegt.

3) Kombination: 11111100
Eingeschaltete Quellen: A,B
Ergebnis: Quellfenster A wird im Zielfenster D abgelegt, ohne den Hintergrund zu zerstören. Wichtig: Quellfenster B muß gleich dem Zielfenster sein, um ein korrektes Ergebnis zu ermöglichen.

3.3. KOPIEREN

3.3.1. SPEICHER LÖSCHEN

Da der Blitter in der Lage ist, mit großer Geschwindigkeit Daten im Speicher umherzuschieben, können wir ihn wunderbar zum Speicherlöschen einsetzen. Dieses kurze Programm zeigt gut, welche Schritte nötig sind, um den Blitter zu initialisieren, und hilft somit, die Arbeitsweise des Blitters zu demonstrieren.
Sehen wir uns ein Beispielprogramm an, das ein Feld in der Größe 320 mal 256 Punkten (320/8*256=10240 Byte) löscht. Diesen Speicherbereich definieren wir am Ende unseres Programms mit der Anweisung

```
pic:        blk.b       10240,$ff
```

$ff bedeutet, daß der reservierte Speicher beim Assemblieren mit dem Hex-Wert $ff gefüllt wird. Das machen wir deshalb, damit wir nach dem Start kontrollieren können, ob unser Programm richtig gearbeitet hat. Als nächstes fragen wir den Blitter, ob er gerade beschäftigt ist, indem wir Bit 14 des Registers $dff002 auslesen:

```
clearpic:   btst        #14,$dff002
            bne.s       clearpic
```

Ist er gerade beim Kopieren von Daten, so verzweigt das Programm in der Schleife und wartet, bis er seine Aufgabe beendet hat. Nächster Schritt ist das Initialisieren des Zielkanals.

```
            move.l      #pic,$dff054
```

Als nächstes werden die Moduli, sowie das Register BLTCON1 initialisiert. In diesem Fall werden sie mit 0 beschrieben.

```
            clr         $dff066
            clr         $dff042
```

Jetzt müssen wir noch den Zielkanal einschalten:

```
        move    #%0000000100000000,$dff040
```

Abschließend wird die Größe des Blitterfensters festgelegt und der Blitter somit gestartet.

```
        move    #[256*64]+[320/16],$dff058
```

Nun können wir das Programm starten. Zur Überprüfung sehen wir uns den Speicher an mit (nur SEKA-Kommando!)

```
        q.l pic
```

Es sollten ab dieser Adresse nur noch Nullen anstatt des Strings $ff stehen. Im Überlick nochmals das gesamte Listing:

```
;
;               Speicher löschen
;               löscht eine 320x256 großes Fenster
;
clearpic:       btst    #14,$dff002
                bne.s   clearpic
;
                move.l  #pic,$dff054
                clr     $dff066
                clr     $dff042
                move    #%0000000100000000,$dff040
                move    #[256*64]+[320/16],$dff058
;
                rts
;
pic:            blk.b   10240,$ff
;
```

Wieso wird aber der Speicher gelöscht, obwohl wir gar keine Quellen eingeschaltet haben? Nun, man könnte genausogut eine

Quelle einschalten und auf einen leeren Speicherbereich zeigen lassen, von wo die Daten geholt werden und anschließend in den Zielbereich übertragen werden. Es gibt aber eine, wie oben demonstriert, viel einfachere und wesentlich schnellere Methode. Durch das Abschalten jeglicher Minitermkombinationen

```
move #%0000000100000000,$dff040
```

ist das vom Blitter ausgeführte Ergebnis der Verknüpfung in jedem Fall 0. Daher erübrigt sich das Einschalten der Quellen.

3.3.2. EINFACHES KOPIEREN

Als nächstes wollen wir eine Routine schreiben, die ein 32 farbiges Bob in einen Bildschirm mit ebenfalls fünf Bitplanes hineinkopiert. Wir werden jede Plane mit einem Blitvorgang kopieren und daher den Blitter fünfmal starten. Diese Methode benötigt viel Zeit, ist aber für den Anfang leichter zu verstehen. Wie man dieses Programm noch kürzer und schneller gestalten kann, wird im Kapitel 6 "Tips & Tricks zum Blitter" verraten. Um nun zu kopieren, lesen wir den Anfang der ersten Plane in a1 ein und addieren nach jedem Blit $2800 hinzu, um an den Beginn der nächsten Plane zu gelangen. Der Wert aus a1 wird in das Zielregister übertragen:

```
label:      move.l    a1,$dff054
```

BLTAMOD und BLTCON1 werden mit 0 initialisiert:

```
blitten:    clr.w     $DFF042
            clr.w     $DFF064
```

Da wir weder das erste noch das letzte Word ausmaskieren (weglassen) wollen, müssen wir sämtliche Bits auf eins setzen.

```
            move.w    #$ffff,$dff044
```

```
        move.w    #$ffff,$dff046
```

Weiterhin werden der Modulo-Wert für die Zielquelle angegeben, sowie der DMA-Kanal für Quelle A und Ziel D eingeschaltet. Außerdem werden alle Miniterms gewählt, welche bei A eine wahre Aussage liefern. Dies sind die Bits LF4 - LF7.

```
        move.w    #%0000100111110000,$DFF040
        move.w    #$24,$DFF066
```

Abschließend berechnen wir noch die Größe des Blitterfensters und starten denselben. Unser Bob ist 32 mal 32 Punkte groß, daher errechnet sich die Fenstergröße wie folgt aus der bekannten Formel:

BLTSIZE = Höhe*64 + Breite in Words
BLTSIZE = 32*64 + 2
BLTSIZE = 2050

Sehen wir uns nun das fertige Programm an, das einen Screen mit 5 Bitplanes erstellt und ein 32-farbiges Bob hineinkopiert. Assemblieren Sie das Programm und laden Sie das fertige Bob von der Programmdiskette mit dem "ri"-Befehl (Seka-Assembler) nach.
Hier nun das fertige Beispielprogramm:

```
;
;    Einfaches Kopieren mit dem Blitter
;
;Nachzuladendes Bild:  df0:raw/BOB32x32_5.raw  auf  blitpic
;
;
s:          bsr.s       opening
            bsr.L       initcolors
            bsr.L       blitten
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$00003000,d0
```

```
            bne.s       loop
            btst        #6,$bfe001
            bne.s       loop
;
;
end:        move.w      #$c000,$dff09a
e:          rts
;
opening: move.w         #$4000,$dff09a
            move.l      #copperl,$dff084
            move.l      #pic,d0
            lea         pl1+2(pc),a0
            moveq.l     #4,d1
initplanes:
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            add.l       size,d0
            dbf         d1,initplanes
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.l     #7,d1
initsprites:
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
```

```
initcolors:
            lea         blitpic+640,a0
            lea         colors(pc),a1
            move.w      #31,d0
            move.w      #$0180,d1
copycolors:
            move.w      d1,(a1)+
            move.w      (a0)+,(a1)+
            add.w       #2,d1
            dbf         d0,copycolors
            rts
;
blitten:    clr.w       $DFF042
            clr.w       $DFF064
;
            move.w      #$ffff,$dff044
            move.w      #$ffff,$dff046
;
            move.w      #%0000100111110000,$DFF040
            move.w      #$24,$DFF066
            move.l      #blitpic,$DFF050
            lea         pic+2016,a1
            moveq.l     #4,d4
;
hier:       move.l      a1,$dff054
            add.l       #10240,a
            move.w      #2050,$DFF058
;
btest2:     btst        #6,$DFF002
            bne.s       btest2
;
            dbf         d4,hier
            rts
;
copperl:
            dc.w        $008e,$2071
```

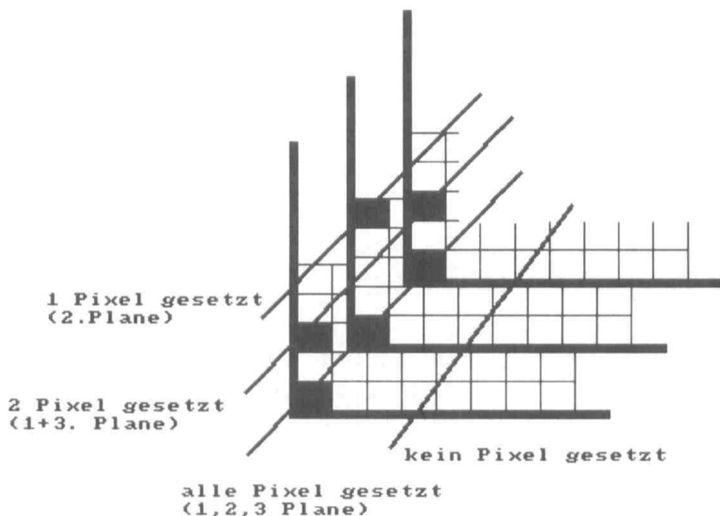
```
        dc.w    $0090,$20d4
        dc.w    $0092,$0038
        dc.w    $0094,$00d0
        dc.w    $0102,$0000
        dc.w    $0104,$0000
        dc.w    $0108,$0000
        dc.w    $010a,$0000
        dc.w    $0100,$5200
pl1:    dc.w    $00e0,$0000
        dc.w    $00e2,$0000
pl2:    dc.w    $00e4,$0000
        dc.w    $00e6,$0000
pl3:    dc.w    $00e8,$0000
        dc.w    $00ea,$0000
pl4:    dc.w    $00ec,$0000
        dc.w    $00ee,$0000
pl5:    dc.w    $00f0,$0000
        dc.w    $00f2,$0000
sp1:    dc.w    $0120,$0000
        dc.w    $0122,$0000
        dc.w    $0124,$0000
        dc.w    $0126,$0000
        dc.w    $0128,$0000
        dc.w    $012a,$0000
        dc.w    $012c,$0000
        dc.w    $012e,$0000
        dc.w    $0130,$0000
        dc.w    $0132,$0000
        dc.w    $0134,$0000
        dc.w    $0136,$0000
        dc.w    $0138,$0000
        dc.w    $013a,$0000
        dc.w    $013c,$0000
        dc.w    $013e,$0000
colors: blk.w   64,0
        dc.w    $ffff,$fffe
```

```
;
sprite:     dc.l      0
size:       dc.l      10240
blitpic:    blk.b     704,0
pic:        blk.b     51200,0
;
```

3.3.3. KOPIEREN MIT MASKE

Als nächstes wollen wir ein 32 Blitterobjekt, auch Bob genannt, welches 5 Bitplanes besitzt (das entspricht 32 Farben) in ein ebenfalls 32-farbiges Hintergrundbild hineinkopieren. Dies soll derart geschehen, daß das hinter dem Bob liegende Bild nicht zerstört wird. Wo liegt hierbei das Problem? Nun, normalerweise kopiert der Blitter das gesamte Objekt, ohne darauf zu achten, was auf der Zielplane liegt (wir haben das in den vorangegangenen Beispielprogrammen so praktiziert). Kommt es nun vor, daß dieses Objekt einige Stellen enthält, an denen die Hintergrundfarbe erscheint, dann sollte dort das Hintergrundbild durchleuchten. Bei normaler Kopierweise sind diese Stellen jedoch schwarz. Sehen wir uns dazu die Abbildung auf der nächsten Seite an:

Das Problem ist folgendes: Damit der Hintergrund an den Stellen durch das Bob durchscheint, an denen auf allen Planes des Bobs die Bits gelöscht sind, muß man ihm mitteilen, wo sich diese Bits befinden.
Der Blitter kann natürlich nicht wissen, welche Bits wirklich auf allen drei Planes gelöscht sind. Es kann ja, wie in unserer Abbildung der Fall sein, daß ein Bit auf der ersten Plane zwar 0 ist, aber dafür auf der zweiten Plane gesetzt wurde, damit eine andere Farbe entsteht.



Abhilfe schafft die sogenannte Maske, die dem Blitter genau mitteilt, welche Bits auf allen Planes benötigt werden. Das bedeutet, man legt dem Blitter mit der Maske eine Vorlage vor, die das Aussehen des Bobs auf allen verwendeten Bitplanes wiederspiegelt. Mit anderen Worten gesagt, muß eine Maske auf jeder Bitplane das gleiche Bitmuster beinhalten, wie das Objekt, das kopiert werden soll. Aus diesem Grund ist die Maske auch einfarbig.
Mit einem Malprogramm wie DPaint erstellt man eine Maske am besten, indem man das Bob mit der letzten Farbe (bei 5 Planes mit Farbe Nummer 31) übermalt. Wir erinnern uns, wenn man mit der letzten verfügbaren Farbe malt, werden logischerweise die Bits auf allen Planes gesetzt, um den Farbton zu erhalten. Die Maske wird anschließend konvertiert und zusätzlich zu unserem Bob während des Blitvorgangs mitverknüpft.

```
move.w #%0000111111000010,BLTCON0(A6)
move.l  #msk,BLTBPTH(a6)
```

Dazu müssen wir mit der Minitermkombination die richtige Verknüpfung anwählen, sowie die zweite Quelle B einschalten, in der die Maske eingelesen wird.

```
;
;               Kopieren mit Maske
;
;               Nachzuladende Files:
;               "raw/Bob32x32_5.raw" auf "bob"
;               "raw/Bob32x32_5.msk" auf "msk"
;               "raw/Hintergrund.raw" auf "pic"
;
dmaconr    =          $02
bltddat    =          $00
bltcon0    =          $40
bltcon1    =          $42
bltafwm    =          $44
bltalwm    =          $46
bltcpth    =          $48
bltbpth    =          $4c
bltapth    =          $50
bltaptl    =          $52
bltdpth    =          $54
bltsize    =          $58
bltcmod    =          $60
bltbmod    =          $62
bltamod    =          $64
bltdmod    =          $66
bltcdat    =          $70
bltbdat    =          $72
bltadat    =          $74
;
s:         bsr.s      opening
           bsr.L      initcolors
;
;
```

```
        move.l   #$DFF000,a6
        bsr.L    Blit                ; Routine anspringen
loop:   btst     #6,$bfe001
        bne.s    loop
;
;
end:    move.w   #$c000,$dff09a
        moveq.l  #0,d0
e:      rts
;
opening:
        move.w   #$4000,$dff09a
        move.l   #copperl,$dff084
        move.l   #pic,d0
        move     d0,pl1+6
        swap     d0
        move     d0,pl1+2
        swap     d0
        add.l    size(pc),d0
        move     d0,pl2+6
        swap     d0
        move     d0,pl2+2
        swap     d0
        add.l    size(pc),d0
        move     d0,pl3+6
        swap     d0
        move     d0,pl3+2
        swap     d0
        add.l    size(pc),d0
        move     d0,pl4+6
        swap     d0
        move     d0,pl4+2
        swap     d0
        add.l    size(pc),d0
        move     d0,pl5+6
        swap     d0
```

```
            move        d0,pl5+2
            swap        d0
;
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.l     #7,d1
initsprites: swap       d0
            move        d0,(a0)
            add.l       #4,a0
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
;
initcolors: lea         color(pc),a0
            lea         pic+40000,a1
            moveq       #31,d0
            move        #$180,d1
copycolors: move        d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
;
            rts
;
copperl     dc.w        $0090,$f8f0
            dc.w        $0092,$0038
            dc.w        $0094,$00d0
            dc.w        $0102,$0000
            dc.w        $0104,$0024
            dc.w        $0108,$0000
            dc.w        $010a,$0000
            dc.w        $0100,$5300
pl1:        dc.w        $00e0,$0000
```

```
            dc.w      $00e2,$0000
pl2:        dc.w      $00e4,$0000
            dc.w      $00e6,$0000
pl3:        dc.w      $00e8,$0000
            dc.w      $00ea,$0000
pl4:        dc.w      $00ec,$0000
            dc.w      $00ee,$0000
pl5:        dc.w      $00f0,$0000
            dc.w      $00f2,$0000
sp1:        dc.w      $0120,$0000
            dc.w      $0122,$0000
            dc.w      $0124,$0000
            dc.w      $0126,$0000
            dc.w      $0128,$0000
            dc.w      $012a,$0000
            dc.w      $012c,$0000
            dc.w      $012e,$0000
            dc.w      $0130,$0000
            dc.w      $0132,$0000
            dc.w      $0134,$0000
            dc.w      $0136,$0000
            dc.w      $0138,$0000
            dc.w      $013a,$0000
            dc.w      $013c,$0000
            dc.w      $013e,$0000
color:      blk.w     64,0
            dc.w      $ffff,$fffe
;
size:       dc.l      8000
;
;*******************************************
;
Blit:
            move.l    #0,d1
            move.l    #0,d2
```

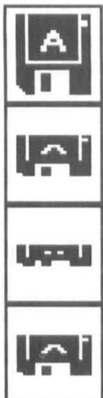
```
NoEnd:
        move.w  #%0000111111000010,BLTCON0(A6)
        clr.w   BLTCON1(a6)
;
        clr.w   BLTAMOD(a6)
        clr.w   BLTBMOD(a6)
        move.w  #36,BLTCMOD(a6)
        move.w  #36,BLTDMOD(a6)
;
        move.w  #$FFFF,BLTAFWM(a6)
        move.w  #$FFFF,BLTALWM(a6)
;
        move.l  #bob,d0
        add.l   d1,d0
        move.l  d0,BLTAPTH(a6)    ;; Adresse d. Bobs
        move.l  #msk,BLTBPTH(a6)
;
        move.l  #pic+2576,d0          ;;Adresse d. Bitplane
                                                   Position
        add.l   d2,d0
        move.l  d0,BLTDPTH(a6)
        move.l  d0,BLTCPTH(a6)
;
        move.w  #2050,BLTSIZE(a6)
;
Wait:   btst    #6,DMACONR(a6)
        bne.s   Wait
;
        add.l   #128,d1
        add.l   #8000,d2
        cmp.l   #40000,d2
        bne.s   NoEnd
        rts

;;************************************************ ;
sprite: dc.l    0
```

```
bob:      blk.b     704,0
msk:      blk.b     640,0
pic:      blk.b     40064,0
;
```

3.3.4. ANIMATIONEN

Abschließend zum Kapitel "Kopieren" wollen wir eine einfache Animation mit Hilfe des Blitters laufen lassen.
Zur Aufgabenstellung: Ein 32 mal 32 Punkte großes Objekt mit einer Bitplane (2 Farben) wird auf dem Bildschirm dargestellt. Von diesem Objekt existieren vier verschiedene Animationsphasen. Das bedeutet, daß die Grafik so gezeichnet wird, daß sich jedes der vier Bildchen geringfügig vom Vorgänger unterscheidet. Legt man diese Bilder nun rasch übereinander, nimmt das Auge eine gleichmäßige Bewegung wahr.
Dieses Objekt kann beispielsweise eine Figur sein, deren Füße sich bewegen. Werden die Bilder übereinander gelegt, so glaubt man, die Figur würde gehen. In unserem Beispiel handelt es sich um eine Diskette, die sich um ihre Achse dreht. Die mit einem Malprogramm gezeichneten Animationsphasen müssen, wie in der Abbildung zu sehen ist, untereinander abgelegt sein.



Anschließend muß man diese Grafik, um sie in unserem Listing zu verwenden, mit einem geeigneten Konvertierprogramm vom IFF- in das RAW-Format umwandeln. In der Unterroutine "Animation" werden die Phasen gewechselt und auf den Bildschirm geblittet.
Animation:

```
move.w   #%0000100111110000,BLTCON0(A6)
```

Zuerst wird das BLTCON0-Register mit der Minitermkombination %11110000, die für Kopieren von A nach D steht, initialisiert. Ebenfalls werden Quelle A und Ziel D eingeschaltet.

```
clr.w    BLTCON1(a6)
clr.w    BLTAMOD(a6)
```

BLTCON1 und BLTAMOD werden auf 0 gesetzt.

```
move.w   #36,BLTDMOD(a6)
```

BLTDMOD hingegen muß auf 36 gesetzt werden, damit der Blitter das 32 Pixel breite Objekt richtig kopiert.

```
move.w   #$FFFF,BLTAFWM(a6)
move.w   #$FFFF,BLTALWM(a6)
```

Das erste und letzte Word werden ausmaskiert.

```
move.l   #bob,d0
move.l   #circletab,a0
add.l    Phase,a0
add.l    (a0),d0
move.l   d0,BLTAPTH(a6)
move.l   #pic,BLTDPTH(a6)
```

Die obigen Zeilen tragen die Bitplane, auf die kopiert werden soll ein, sowie diejenige Animationsphase, die als nächstes an der Reihe

ist. Welche Phasen in welcher Reihenfolge geblittet werden, ist im "circletab" festgelegt:

```
circletab:
            dc.l        000,000,000,000,000,000
            dc.l        128,128,128,128,128,128
            dc.l        256,256,256,256,256,256
            dc.l        384,384,384,384,384,384
            dc.l        256,256,256,256,256,256
            dc.l        128,128,128,128,128,128
```

Jede Zahl, die in dieser Tabelle abgelegt ist, wird zur Adresse, an der die vier Animationsphasen liegen, hinzuaddiert. Da eine Phase 128 Byte (32/8*32=128) lang ist, liegt die erste bei 0, die zweite bei 128, die dritte bei 256, u.s.w.

```
            move.w  #2050,BLTSIZE(a6)
```

Als letztes wird die aktuelle Phase geblittet, indem der Blitvorgang durch Beschreiben des BLTSIZE-Registers gestartet wird. Die BLTSIZE errechnet sich wie folgt: 32*64+2=2050

```
;
;           ANIMATIONEN
;
;           Nachzuladendes File:
;
;           "raw/Bob32x32_1pl.raw"  auf  "Bob"
;
;
dmaconr     =       $02
bltddat     =       $00
bltcon0     =       $40
bltcon1     =       $42
bltafwm     =       $44
bltalwm     =       $46
bltcpth     =       $48
```

```
bltbpth     =           $4c
bltapth     =           $50
bltaptl     =           $52
bltdpth     =           $54
bltsize     =           $58
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
;
s:          bsr.s       opening
            bsr.L       initcolors
;
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$08000,d0
            bne.s       loop
;
            move.l      #$dff000,a6
            bsr.L       Animation
;
            btst        #6,$bfe001
            bne.s       loop
;
;
end:        move.w      #$c000,$dff09a
            moveq.l     #0,d0
e:          rts
;
opening:    move.w      #$4000,$dff09a
            move.l      #copperl,$dff084
```

```
            move.l    #pic,d0
            move      d0,pl1+6
            swap      d0
            move      d0,pl1+2
            swap      d0
;
            move.l    #sprite,d0
            lea       sp1+2(pc),a0
            moveq.l   #7,d1         ;8 Sprites
initsprites: swap     d0
            move      d0,(a0)
            add.l     #4,a0
            swap      d0
            move      d0,(a0)
            add.l     #4,a0
            dbf       d1,initsprites
            rts
;
;
initcolors: lea       color(pc),a0
            lea       colortable(pc),a1
            moveq     #1,d0
            move      #$180,d1
copycolors:
            move      d1,(a0)+
            move      (a1)+,(a0)+
            addq      #2,d1
            dbf       d0,copycolors
            rts
;
colortable: dc.w      $0000,$0fff
;
copperl:
            dc.w      $008e,$3090
            dc.w      $0090,$f8f0
            dc.w      $0092,$0038
```

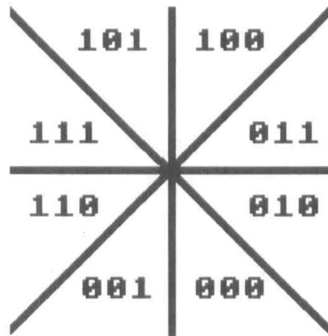
```
        dc.w            $0094,$00d0
        dc.w            $0102,$0000
        dc.w            $0104,$0024
        dc.w            $0108,$0000
        dc.w            $010a,$0000
        dc.w            $0100,$1200
pl1:    dc.w            $00e0,$0000
        dc.w            $00e2,$0000
color:  blk.b           8,0
sp1:    dc.w            $0120,$0000
        dc.w            $0122,$0000
        dc.w            $0124,$0000
        dc.w            $0126,$0000
        dc.w            $0128,$0000
        dc.w            $012a,$0000
        dc.w            $012c,$0000
        dc.w            $012e,$0000
        dc.w            $0130,$0000
        dc.w            $0132,$0000
        dc.w            $0134,$0000
        dc.w            $0136,$0000
        dc.w            $0138,$0000
        dc.w            $013a,$0000
        dc.w            $013c,$0000
        dc.w            $013e,$0000
                dc.w            $ffff,$fffe
;
Animation:
        move.w  #%0000100111110000,BLTCON0(A6)
        clr.w   BLTCON1(a6)
;
        clr.w   BLTAMOD(a6)
        move.w  #36,BLTDMOD(a6)
;
        move.w  #$FFFF,BLTAFWM(a6)
        move.w  #$FFFF,BLTALWM(a6)
```

```
;
            move.l      #bob,d0
            move.l      #circletab,a0
            add.l       Phase,a0
            add.l       (a0),d0
            move.l      d0,BLTAPTH(a6)
;
            move.l      #pic,BLTDPTH(a6) ;
            move.w      #2050,BLTSIZE(a6) ;
            add.l       #4,phase
            cmp.l       #24*6,phase
            bne.s       weiter
            move.l    #0,phase
;
weiter:     rts
;
sprite:     dc.         l 0
phase:      dc.         l 0
;
circletab:
            dc.l        000,000,000,000,000,000
            dc.l        128,128,128,128,128,128
            dc.l        256,256,256,256,256,256
            dc.l        384,384,384,384,384,384
            dc.l        256,256,256,256,256,256
            dc.l        128,128,128,128,128,128
;
bob:        blk.b       512,0
pic:        blk.b       8000,0
;
```

3.4. DER LINE-BEFEHL

3.4.1. WIE ZEICHNET MAN LINIEN?

Neben dem normalen Kopier-Modus verfügt der Blitter über einen Linien-Modus, mit dem es möglich ist, Linien schneller zu zeichnen, als dies mit dem Prozessor möglich wäre.
Was ist aber eigentlich eine Linie? Eine Linie ist eine Verbindung zwischen zwei Punkten, die dadurch entsteht, daß mehrere Punkte aneinander gereiht werden.
Nun leider ist es nicht so ohne weiteres möglich, dem Blitter die Koordinaten für eine Linie zu übergeben und ihn diese zeichnen zu lassen, denn die verschiedenen Blitterregister, die wir schon kennengelernt haben, werden teilweise im Linien-Modus anders verwendet.
Der Hauptschalter, der den Linien-Modus aktiviert, ist im Register BLTCON0 in Bit 0 zu finden. Dieses Bit muß auf 1 gesetzt werden, damit der Linien-Modus aktiviert wird.
In den Registern BLTCPT und BLTDPT muß die Adresse jenes Words der Bitplane liegen, in dem sich der Startpunkt der Linie befindet.
Welcher der 16 Punkte der genaue Startpunkt der Linie ist, wird in die Bits 12 bis 15 von BLTCON0 eingetragen. Ebenfalls sind die USEx-Bits (Bits 8-11) auf %1011 zu setzen.
Das war ja bis jetzt noch relativ einfach, nur leider ist die Berechnung des Endpunktes etwas komplizierter. Wir müssen zuerst angeben, in welche Richtung die Linie führt (siehe Abbildung auf der nächsten Seite).
Je nachdem, in welchen Abschnitt die Linie führt, muß die entsprechende Bitkombination mit den Bits 2 bis 4 des Registers BLTCON1 angegeben werden.



Nun wird die Steigung berechnet. Dazu benötigen wir den Startpunkt (X0,Y0) und den Endpunkt (X1,Y1), sowie den zugehörigen Delta-Wert.

DeltaX = |X0-X1|
DeltaY = |Y0-Y1|

Nun vergleichen wir beide Delta-Werte miteinander und nennen den kleineren DX und den größeren DY. Wenn 2*DX < DY ist, muß Bit 6 von BLTCON1 auf 1 gesetzt werden, ansonsten auf 0. DY ist gleichzeitig die Länge der Linie, wobei Blitterlinien maximal 1024 Punkte lang sein können. Soll eine Linie die maximale Ausdehnung haben, so muß DY auf 0 gesetzt sein. Normalerweise berechnet sich die BLTSIZE: Höhe*64 + Breite in Words. Im Linien-Modus ist dies auch etwas anders: BLTSIZE = Länge der Linie*64 + 2. Im Register BLTADAT muß Bit 15 gesetzt werden.

In die LFx-Bits des Registers BLTCON0 muß die richtige Miniterm-Kombination geschrieben werden. In unserem Fall ist das der Wert %11001010.

BLTAPTL wird mit 2*DX-DY beschrieben. In BLTAMOD folgt 2*(DX-DY) und in BLTBMOD 2*DX. Die Modulo-Register BLTCMOD und BLTDMOD werden mit der Breite des Bildes in Bytes adressiert.

Das Masken-Register BLTAFWM muß ausmaskiert werden, das heißt, daß alle Bits gesetzt werden müssen.

In das Register BLTBDAT kommt das Muster (auch Maske genannt), mit dem die Linie gezeichnet werden soll. Denn der Blitter ist in der Lage, wordweise ein Muster in die Linie zu zeichnen. Will man eine durchgehende Linie, so ist $ffff zu verwenden. Die Maske der Linie kann innerhalb eines Words auch noch pixelweise verschoben werden. Will man diese verschieben, so ist dies mit den Bits 12 bis 15 von BLTCON1 zu tun.

Wichtig für das Flächenfüllen, das wir einige Seiten weiter hinten ausführlich besprechen werden, ist eine exakte Begrenzung, das bedeutet, es darf immer nur ein Punkt in einer Zeile sein. Der Blitter unterstützt dies durch eine spezielle Funktion. Setzt man Bit 1 von BLTCON1 auf 1, dann wird nur 1 Pixel pro Zeile gesetzt.

Ein fertiges Beispielprogramm, das Ihnen die aufwendigen Berechnungen des Start- und Endpunktes abnimmt, finden Sie auf den nächsten Seiten.

3.4.2. GEMUSTERTE LINIEN

Es folgt nun das fertige Beispiellisting, das die im vorhergehenden Kapitel besprochenen Linien mit Hilfe des Blitters auf den Bildschirm zeichnet. Die Routine ist so gehalten, daß man ihr nur die Bitplaneadresse, die Koordinaten des Anfangs- und Endpunktes, sowie die Hardwarebasisadresse übergeben muß.

```
;****DrawLine ********************************************
;*
;*          A4: Zeiger auf Bitplane
;*          A6: Basisadresse $DFF000
;*          D0.w: x0
;*          D1.w: y0
```

```
;*          D2.w: x1
;*          D3.w: y1
;*
;*****************************************************
```

Das Programm errechnet sämtliche anderen notwendigen Parameter, die sich daraus ergeben. Da man mit dem Linien-Modus den Linien ein beliebiges Muster geben kann, wird auch dieses vor dem Unterroutinenaufruf übergeben.

```
        move.w  #$ffff,bltbdat(a6)      ;Muster d. Linie
```

Wird $ffff übergeben, so sind alle Bits gesetzt und die Linie wird durchgehend gezeichnet. In unserem Beispiellisting werden zwei Linien gezeichnet, wobei die eine durchgehend und die andere gemustert dargestellt wird.

```
;
;       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       % LINIEN ZIEHEN MIT DEM BLITTER  %
;       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
;
bltbmod =       $62
bltaptl =       $52
bltcon1 =       $42
bltamod =       $64
bltadat =       $74
bltbdat =       $72
bltafwm =       $44
bltcmod =       $60
bltdmod =       $66
bltcon0 =       $40
bltcpth =       $48
bltdpth =       $54
bltsize =       $58
```

```
dmaconr =       $02
;
;
s:      bsr.L   opening
        bsr.L   initcolors
;
;
        move.w  #0,d0               ;X1-Koordinate
        move.w  #0,d1               ;Y1-Koordinate
        move.w  #150,d2             ;X2-Koordinate
        move.w  #100,d3             ;Y2-Koordinate
        move.l  #pic,a4             ;Adresse der Bitplane
        move.l  #$dff000,a6         ;Basisadresse
        move.w  #$f0f0,bltbdat(a6)  ;Muster d. Linie
        bsr.L   Drawline            ;Routine anspringen
;
WaitLoop2
        :btst   #6,DMACONR(A6)  ;Blitterwait
        bne.s   WaitLoop2
;
        move.w  #150,d0             ;X1-Koordinate
        move.w  #100,d1             ;Y1-Koordinate
        move.w  #320,d2             ;X2-Koordinate
        move.w  #200,d3             ;Y2-Koordinate
        move.l  #pic,a4             ;Adresse der Bitplane
        move.l  #$dff000,a6         ;Basisadresse
        move.w  #$ffff,bltbdat(a6)  ;Muster d. Linie
        bsr.L   Drawline            ;Routine anspringen
;
loop:   move.l  $dff004,d0
        and.l   #$fff00,d0
        cmp.l   #$00003000,d0
        bne.s   loop
;
        btst    #6,$bfe001
        bne.s   loop
```

```
;
;
end:        move.w   #$c000,$dff09a
            moveq.l  #0,d0
e:          rts
;
opening:    move.w   #$4000,$dff09a
            move.l   #copperlist,$dff084
            move.l   #pic,d0
            move     d0,pl1+6
            swap     d0
            move     d0,pl1+2
            swap     d0
;
            move.l   #sprite,d0
            lea      sp1+2(pc),a0
            moveq.l  #7,d1                    ;8 Sprites
initsprites: swap    d0
            move     d0,(a0)
            add.l    #4,a0
            swap     d0
            move     d0,(a0)
            add.l    #4,a0
            dbf      d1,initsprites
            rts
initcolors: lea      color(pc),a0
            lea      colortable(pc),a1
            moveq    #1,d0
            move     #$180,d1
copycolors:  move    d1,(a0)+
            move     (a1)+,(a0)+
            addq     #2,d1
            dbf      d0,copycolors
            rts
;
;*****DrawLine****************************************
```

```
;*
;*      A4: Zeiger auf Bitplane
;*      A6: Basisadresse $DFF000
;*      D0.w: x0
;*      D1.w: y0
;*      D2.w: x1
;*      D3.w: y1
;*
;*****************************************************
;
DrawLine: movem.l   D0-D5/A4,-(A7)
;
          sub.w     D0,D2
          bmi.s     x0gx1
          sub.w     D1,D3
          bmi.s     y0gy10
          cmp.w     D3,D2
          bmi.s     dygdx0
          moveq     #[4*4]!1,D4
          bra.s     Ok0
;
dygdx0:   exg       D3,D2
          moveq     #1,D4
          bra.s     Ok0
;
y0gy10:   neg.w     D3
          cmp.w     D3,D2
          bmi.s     dygdx1
          moveq     #[6*4]!1,D4
          bra.s     Ok0
;
dygdx1:   exg       D3,D2
          moveq     #[1*4]!1,D4
          bra.s     Ok0
;
x0gx1:    neg.w     D2
```

```
            sub.w       D1,D3
            bmi.s       y0gy11
            cmp.w       D3,D2
            bmi.s       dygdx2
            moveq       #[5*4]!1,D4
            bra.s       Ok0
;
dygdx2:     exg         D3,D2
            moveq       #[2*4]!1,D4
            bra.s       Ok0
;
y0gy11:     neg.w       D3
            cmp.w       D3,D2
            bmi.s       dygdx3
            moveq       #[7*4]!1,D4
            bra.s       Ok0
;
dygdx3:     exg         D3,D2
            moveq       #[3*4]!1,D4
;
Ok0:        lsl.w       #2,D3
;
WaitLoop: btst          #6,DMACONR(A6)
            bne.s       WaitLoop
;
            move.w D3,BLTBMOD(A6)
;
            add.w       D2,D2
            sub.w       D2,D3
            bpl.s       Ok1
            or.b        #%1000000,D4
Ok1:        move.w      D3,BLTAPTL(A6)
            or.w        #%1111000000000000,D4
            move.w      D4,BLTCON1(A6)
;
            sub.w       D2,D3
```

```
        move.w   D3,BLTAMOD(A6)
;
        move.w   #$8000,BLTADAT(A6)
        moveq    #-1,D3
        move.l   D3,BLTAFWM(A6)
        move.w   #40,BLTCMOD(A6)
        move.w   #40,BLTDMOD(A6)
;
        move.w   D0,D3
        and.w    #$f,D3
        ror.w    #4,D3
        or.w     #$bea,D3
        move.w   D3,BLTCON0(A6)
;
        lsr.w    #3,D0
        mulu     #40,D1
        add.w    D1,D0
        lea.l    (A4,D0.w),A4
        move.l   A4,BLTCPTH(A6)
        move.l   A4,BLTDPTH(A6)
;
        lsl.w    #5,D2
        add.w    #$42,D2
        move.w   D2,BLTSIZE(A6)
;
        movem.l  (A7)+,D0-D5/A4 rts
;
copperlist:
        dc.w     $008e,$3090
        dc.w     $0090,$f8f0
        dc.w     $0092,$0038
        dc.w     $0094,$00d0
        dc.w     $0102,$0000
        dc.w     $0104,$0024
        dc.w     $0108,$0000
        dc.w     $010a,$0000
```

```
            dc.w      $0100,$1200
pl1:        dc.w      $00e0,$0000
            dc.w      $00e2,$0000
sp1:        dc.w      $0120,$0000
            dc.w      $0122,$0000
            dc.w      $0124,$0000
            dc.w      $0126,$0000
            dc.w      $0128,$0000
            dc.w      $012a,$0000
            dc.w      $012c,$0000
            dc.w      $012e,$0000
            dc.w      $0130,$0000
            dc.w      $0132,$0000
            dc.w      $0134,$0000
            dc.w      $0136,$0000
            dc.w      $0138,$0000
            dc.w      $013a,$0000
            dc.w      $013c,$0000
            dc.w      $013e,$0000
color:      blk.w     4,0
            dc.w      $ffff,$fffe
;
colortable: dc.w      $0000,$0fff
sprite:     dc.       l 0
;
pic:        blk.b     8000,0
;
```

3.4.3. OBJEKTE DREHEN

Wir haben gelernt, daß man mit dem Blitter blitzschnell Linien ziehen kann, und diese auch ein Muster enthalten dürfen. Diese Fähigkeit machen wir uns zunutze, um ein Objekt zu drehen.

Wie funktioniert das im Detail? Wir haben ein Objekt, das 16 Punkte breit und 15 Zeilen hoch ist. Wir projizieren jetzt jede horizontale Linie des Objektes auf eine Linie. Das bedeutet, wir haben jetzt 15 Linien, die je 16 Pixel breit sind und jede Linie hat als Muster die Bitkombination, die die jeweilige Zeile unseres Objektes besitzt.

Wir wissen auch, daß wir eine Blitter-Linie in jedem beliebigen Winkel darstellen können. Und genau das machen wir jetzt auch mit unserem Objekt. Wir drehen alle 15 Zeilen, indem wir den richtigen Winkel berechnen. Der Prozessor stellt uns leider keine Sinus/Cosinus-Routine zur Verfügung, daher müssen wir selber eine schreiben. Nur würde eine Echtzeitberechnung zu lange dauern, daher sind die Sinus-Werte in einer Tabelle abgelegt und werden von der folgenden Routine verwendet.

```
cos:      add.b     #64,d6
Sin:      move.b    D6,D7
          and.w     #$ff,D6
          add.w     D6,D6
          and.w     #$ff,D6
          move.w    SinTable(PC,D6.w),D6
          btst      #7,D7
          beq.s     Return
          neg.w     D6
Return:   rts
```

Ein Problem tritt noch auf, denn, wenn das Objekt gedreht wird, kommt es bei einer ungünstigen Winkellage vor, daß zwischen den Linien der Hintergrund durchscheint. Diesen Schönheitsfehler kann man leicht dadurch beheben, daß man eine zweite Bitplane ebenfalls

mit derselben Adresse adressiert, wie sie die erste Bitplane hat. Diese zweite Bitplane wird um eine Zeile nach oben verschoben. Durch die Überdeckung werden die unschönen Löcher verdeckt und schon dreht sich unser Objekt sauber!

```
;
;               %%%%%%%%%%%%%%%%
;               % OBJEKTE DREHEN %
;               %%%%%%%%%%%%%%%%
;
;
dmaconr     =           $02
bltddat     =           $00
bltcon0     =           $40
bltcon1     =           $42
bltafwm     =           $44
bltalwm     =           $46
bltcpth     =           $48
bltaptl     =           $52
bltdpth     =           $54
bltsize     =           $58
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
;
s:          bsr.s       opening
            bsr.L       initcolors
;
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$08000,d0
```

```
            bne.s       loop
;
            move.l      #$DFF000,a6
            bsr.L       Drehen
;
            btst        #6,$bfe001
            bne.s       loop
;
;
end:        move.w      #$c000,$dff09a
            moveq.l     #0,d0
e:          rts
;
opening: move.w         #$4000,$dff09a
            move.l      #copperl,$dff084
            move.l      #pic,d0
            move        d0,pl1+6
            swap        d0
            move        d0,pl1+2
            swap        d0
            move        d0,pl2+6
            swap        d0
            move        d0,pl2+2
;
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.l     #7,d1                   ;8 Sprites
initsprites: swap       d0
            move        d0,(a0)
            add.l       #4,a0
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
```

```
initcolors:lea          color(pc),a0
           lea          colortable(pc),a1
           moveq        #3,d0
           move         #$180,d1
copycolors:move         d1,(a0)+
           move         (a1)+,(a0)+
           addq         #2,d1
           dbf          d0,copycolors
;
           rts
;
colortable: dc.w        $0000,$0fff,$0fff,$0fff
;
copperl:
           dc.w         $008e,$3090
           dc.w         $0090,$f8f0
           dc.w         $0092,$0038
           dc.w         $0094,$00d0
           dc.w         $0102,$0010        ;2.Plane 1 Pixel nach rechts
           dc.w         $0104,$0024
           dc.w         $0108,$0000
           dc.w         $010a,$0000
           dc.w         $0100,$2200
pl1:       dc.w         $00e0,$0000
           dc.w         $00e2,$0000
pl2:       dc.w         $00e4,$0000
           dc.w         $00e6,$0000
color:     blk.b        16,0
sp1:       dc.w         $0120,$0000
           dc.w         $0122,$0000
           dc.w         $0124,$0000
           dc.w         $0126,$0000
           dc.w         $0128,$0000
           dc.w         $012a,$0000
           dc.w         $012c,$0000
           dc.w         $012e,$0000
```

```
          dc.w      $0130,$0000
          dc.w      $0132,$0000
          dc.w      $0134,$0000
          dc.w      $0136,$0000
          dc.w      $0138,$0000
          dc.w      $013a,$0000
          dc.w      $013c,$0000
          dc.w      $013e,$0000
          dc.w      $ffff,$fffe
;
;
;*****DrawLine****************************************
;*
;*        A4: Zeiger auf Bitplane
;*        D0: x0
;*        D1: y0
;*        D2: x1
;*        D3: y1
;*
;*****************************************************
DrawLine: movem.l   D0-D5/A4,-(A7)
;
          sub.w     D0,D2
          bmi.s     x0gx1
          sub.w     D1,D3
          bmi.s     y0gy10
          cmp.w     D3,D2
          bmi.s     dygdx0
          moveq     #[4*4]!1,D4
          bra.s     Ok0
;
dygdx0:exg          D3,D2
          moveq     #1,D4
          bra.s     Ok0
;
y0gy10: neg.w       D3
```

```
            cmp.w     D3,D2
            bmi.s     dygdx1
            moveq     #[6*4]!1,D4
            bra.s     Ok0
;
dygdx1: exg           D3,D2
            moveq     #[1*4]!1,D4
            bra.s     Ok0
;
x0gx1:      neg.w     D2
            sub.w     D1,D3
            bmi.s     y0gy11
            cmp.w     D3,D2
            bmi.s     dygdx2
            moveq     #[5*4]!1,D4
            bra.s     Ok0
;
dygdx2: exg           D3,D2
            moveq     #[2*4]!1,D4
            bra.s     Ok0
;
y0gy11: neg.w         D3
            cmp.w     D3,D2
            bmi.s     dygdx3
            moveq     #[7*4]!1,D4
            bra.s     Ok0
;
dygdx3: exg           D3,D2
            moveq     #[3*4]!1,D4
;
Ok0:        lsl.w     #2,D3
;
WaitLoop1:btst        #6,DMACONR(A6)
            bne.s     WaitLoop1
;
            move.w    D3,BLTBMOD(A6)
```

```
;
        add.w     D2,D2
        sub.w     D2,D3
        bpl.s     Ok1
        or.b      #%1000000,D4
Ok1:    move.w    D3,BLTAPTL(A6)
        or.w      #%1111000000000000,D4
        move.w    D4,BLTCON1(A6)
;
        sub.w     D2,D3
        move.w    D3,BLTAMOD(A6)
;
        move.w    #$8000,BLTADAT(A6) moveq  #-1,D3
        move.l    D3,BLTAFWM(A6)
        move.w    #40,BLTCMOD(A6)
        move.w    #40,BLTDMOD(A6)
;
        move.w    D0,D3
        and.w     #$f,D3
        ror.w     #4,D3
        or.w      #$bea,D3
        move.w    D3,BLTCON0(A6)
;
        lsr.w     #3,D0
        mulu      #40,D1
        add.w     D1,D0
        lea.l     (A4,D0.w),A4
        move.l    A4,BLTCPTH(A6)
        move.l    A4,BLTDPTH(A6)
;
        lsl.w     #5,D2
        add.w     #$42,D2
        move.w    D2,BLTSIZE(A6)
;
        movem.l   (A7)+,D0-D5/A4
        rts
```

```
;
;************************************
; Drehen:
;
; Löschen des Bildschirm an der entsprechenden Stelle
;
; Der Einfachkeit halber wird der Bildschirm gleich über
; die ganze Breite gelöscht.
;
           clr.w      BLTDDAT(a6)
           clr.w      BLTCDAT(a6)
           clr.w      BLTBDAT(a6)
           clr.w      BLTADAT(a6)
;
           move.w     #%0000000111110000,BLTCON0(a6)
           clr.w      BLTCON1(a6)
;
           clr.w      BLTAMOD(a6)
           clr.w      BLTDMOD(a6)
;
           move.w     #$FFFF,BLTAFWM(a6)
           move.w     #$FFFF,BLTALWM(a6)
;
           move.l     #pic,d0
           add.l      #1160,d0
           move.l     d0,BLTDPTH(a6)
;
           move.w     #2708,BLTSIZE(a6) ;42*64 + 20
;
WaitLoop2:btst        #6,DMACONR(A6)
           bne.s      WaitLoop2
;
; Berechnung und Zeichnung der Drehung
;
           move.l     #pic,a4
           move.l     #bob,A0
```

```
;
        moveq   #0,D0
        moveq   #0,D1
        moveq   #15,D2
        moveq   #0,D3
;
        move.b  winkel,D6
        bsr.L   Sin
        move.w  D6,D7
;
        move.b  winkel,D6
        move.w  D7,-(A7)
        bsr.L   Cos
        move.w  (A7)+,D7
;
loop2:  movem.w D0-D3,-(A7)
;
        move.w  D0,D4
        move.w  D1,D5
;
        muls    D6,D0
        muls    D7,D5
        add.l   D5,D0
        asr.l   #8,D0
        asr.l   #6,D0
;
        muls    D6,D1
        muls    D7,D4
        sub.l   D4,D1
        asr.l   #8,D1
        asr.l   #6,D1
;
        move.w  D2,D4
        move.w  D3,D5
;
        muls    D6,D2
```

```
        muls     D7,D5
        add.l    D5,D2
        asr.l    #8,D2
        asr.l    #6,D2
;
        muls     D6,D3
        muls     D7,D4
        sub.l    D4,D3
        asr.l    #8,D3
        asr.l    #6,D3
;
        add.l    #120,D0            ;Berechnet wird die
                                    ;Drehung fürdie
        add.l    #50,D1             ;Position (0/0), daher
                                    ;wird eineVerschiebung
        add.l    #120,d2            ;in Richtung  Bild-
                                    ;schirmmitte
        add.l    #50,d3             ;notwendig.
        move.w   (a0)+,BLTBDAT(A6)
        bsr.L    DrawLine
;
        movem.w  (A7)+,D0-D3
        addq.w   #1,D1
        move.w   D1,D3
        cmp.w    #15,D1             ;Zeilenanzahl
        bne.s    Loop2
;
        add.b    #2,winkel          ;Kein Zählen nötig, da
                                    ;die Sinus-Unterroutine
                                    ;nür bis 255 geht.
        rts
;
;**** Sin********************************************
;*
;*      D6: Winkel (0-255)
;*
```

```
;*************************************************
;
cos:      add.b     #64,d6
Sin:      move.b    D6,D7
          and.w     #$ff,D6
          add.w     D6,D6
          and.w     #$ff,D6
          move.w    SinTable(PC,D6.w),D6
          btst      #7,D7
          beq.s     Return
          neg.w     D6
Return:   rts
;
SinTable:
dc.w      $0000,$0192,$0324,$04b5,$0646,$07d6,$0964,$0af1
dc.w      $0c7c,$0e06,$0f8d,$1112,$1294,$1413,$1590,$1709
dc.w      $187e,$19ef,$1b5d,$1cc6,$1e2b,$1f8c,$20e7,$223d
dc.w      $238e,$24da,$2620,$2760,$289a,$29ce,$2afb,$2c21
dc.w      $2d41,$2e5a,$2f6c,$3076,$3179,$3274,$3368,$3453
dc.w      $3537,$3612,$36e5,$37b0,$3871,$392b,$39db,$3a82
dc.w      $3b21,$3bb6,$3c42,$3cc5,$3d3f,$3daf,$3e15,$3e72
dc.w      $3ec5,$3f0f,$3f4f,$3f85,$3fb1,$3fd4,$3fec,$3ffb
dc.w      $4000,$3ffb,$3fec,$3fd4,$3fb1,$3f85,$3f4f,$3f0f
dc.w      $3ec5,$3e72,$3e15,$3daf,$3d3f,$3cc5,$3c42,$3bb6
dc.w      $3b21,$3a82,$39db,$392b,$3871,$37b0,$36e5,$3612
dc.w      $3537,$3453,$3368,$3274,$3179,$3076,$2f6c,$2e5a
dc.w      $2d41,$2c21,$2afb,$29ce,$289a,$2760,$2620,$24da
dc.w      $238e,$223d,$20e7,$1f8c,$1e2b,$1cc6,$1b5d,$19ef
dc.w      $187e,$1709,$1590,$1413,$1294,$1112,$0f8d,$0e06
dc.w      $0c7c,$0af1,$0964,$07d6,$0646,$04b5,$0324,$0192
;
winkel:   dc.w 0
sprite:   dc.l 0
size:     dc.l 8000
```

```
bob:
        dc.w    %1111111111111111
        dc.w    %0000000000000000
        dc.w    %0000000110000000
        dc.w    %0000000110000000
        dc.w    %0001100110011000
        dc.w    %0001100110011000
        dc.w    %0001111111111000
        dc.w    %0001111111111000
        dc.w    %0001111111111000
        dc.w    %0001100110011000
        dc.w    %0001100110011000
        dc.w    %0000000110000000
        dc.w    %0000000110000000
        dc.w    %0000000000000000
        dc.w    %1111111111111111
;
pic:    blk.b   8000,0
```

3.5. FLÄCHEN FÜLLEN

3.5.1. EINFACHES FLÄCHENFÜLLEN

Die dritte und letzte Aufgabe, die der Blitter zusätzlich zum kopieren b.z.w. Linien ziehen ausführen kann, ist das Füllen von Flächen. In den diversen Malprogrammen kann man immer wieder die tollsten Füllfunktionen beobachten. Die Grund-Füll-Funktion des Blitters ist hingegen nicht so komplex.
Damit der Blitter eine Fläche korrekt ausfüllen kann, muß diese sowohl auf der linken, als auch auf der rechten Seite durch ein gesetztes Bit begrenzt sein. Ist dies der Fall, füllt er den Zwischenraum aus, indem er die gelöschten Bits auf 1 setzt.

Bitplane vor der Fülloperation:

```
00000000000100000000100000000000
00000000010000000000001000000000
00000001000000000000000010000000
00000100000000000000000000100000
00010000000000000000000000001000
```

Und nach der Fülloperation:

```
00000000000111111111100000000000
00000000011111111111111000000000
00000001111111111111111110000000
00000111111111111111111111100000
00011111111111111111111111111000
```

Wichtig ist, daß nur ein Bit sowohl links als auch rechts als Begrenzung eingesetzt wird. Denn sind mehrere Bits vorhanden, kommt der Füllalgorithmus des Blitters durcheinander. Dies geschieht deshalb, da der Blitter immer rechts beginnend die Bits testet, bis er auf ein gesetztes Bit trifft. Ist dies der Fall, setzt er das FillCarry-Bit auf 1 und setzt gleichzeitig das Bit in der Grafik. Dieser Vorgang wiederholt sich solange, bis er wieder auf ein gesetztes Bit trifft. Ist nun eine ungerade Anzahl an Begrenzungsbits vorhanden, ist es leicht verständlich, daß der Blitter dadurch irritiert wird.

Normalerweise ist das FC-Bit beim Start der Fülloperation auf Null gesetzt. Bit 2 im BLTCON1-Register ist das FCI (Fill Carry In) Bit und beinhaltet jenen Wert, den das Fill-Carry-Bit am Anfang der Blitteroperation annehmen soll. Setzen wir nun das FCI-Bit auf eins, so beginnt der Blitter schon ganz rechts mit dem Füllen der Flächen und läßt jenen Bereich aus, der innerhalb der Begrenzungslinien liegt.

Die Bitplane vor der Fülloperation mit den Begrenzungslinien:

```
00000000000100000000100000000000
00000000010000000000001000000000
00000001000000000000000010000000
00000100000000000000000000100000
00010000000000000000000000001000
```

Die Bitplane nach der Fülloperation, wenn das Fill-Carry-In Bit (Bit 2 in BLTCON1) auf 0 gesetzt ist.

```
00000000000111111111100000000000
00000000011111111111111000000000
00000001111111111111111110000000
00000111111111111111111111100000
00011111111111111111111111111000
```

Die Bitplane nach der Fülloperation, wenn das Fill-Carry-Bit auf 1 gesetzt ist.

```
11111111111100000000111111111111
11111111110000000000001111111111
11111111000000000000000011111111
11111100000000000000000000111111
11110000000000000000000000001111
```

Da immer zwei Begrenzungs-Bits benötigt werden, um eine Fläche zu füllen, wäre die kleinste Einheit einer Fläche 2 Bits breit. Eine Fläche die nur ein Bit breit ist, wäre unmöglich in einer Fülloperation zu verwenden, da sie immer ein falsches Ergebnis liefern würde. Aus diesem Grund gibt es einen zweiten Füll-Modus. Der erste, den wir oben erläutert haben, bei dem man zwei Begrenzungs-Bits benötigt, ist der sogenannte IFE-Modus (Inclusive Fill Enable). Der zweite Modus ist der EFE-Modus (Exclusive Fill Enable), der ebenfalls zwei Begrenzungs-Bits benötigt. Der Unterschied zum ersten Modus ist, daß der EFE-Modus beim Füllen

immer das linke Begrenzungs-Bit löscht. Es wird also immer jenes Bit gelöscht, wenn das FC-Bit wieder von 1 auf 0 wechselt.

Der Blitter kann das Füllen von Flächen zusätzlich zu einer gestarteten Kopieraktion ausführen und er wird dadurch nicht, wie oftmals fälschlicherweise behauptet, langsamer. Wie wird der Füllmodus nun aber aktiviert?

```
fill:   move.w  #$FFFF,BLTDDAT(a6)
        move.w  #$FFFF,BLTCDAT(a6)
        move.w  #$FFFF,BLTBDAT(a6)
        move.w  #$FFFF,BLTADAT(a6)
;
        clr.w   BLTAMOD(a6)
        clr.w   BLTDMOD(a6)
        move.w  #$FFFF,BLTAFWM(a6)
        move.w  #$FFFF,BLTALWM(a6)
;
        move.w  #%0000100111110000,BLTCON0(a6)
```

Die Blitterregister sollten so initialisiert werden, wie es bei einem normalen Kopiervorgang üblich ist. Das Ergebnis der Verknüpfung der drei Quellen muß durch die Miniterm-Kombination und die Modulo-Werte ein Ergebnis mit korrekten Begrenzungsbits im Ziel D liefern.

```
        move.w  #%0000000000001010,BLTCON1(a6)
```

Als nächstes ist einer der beiden Betriebsmodi für das Füllen zu wählen, dies ist entweden IFE oder EFE (in unserem Fall IFE). Ebenso wird der Descending-Modus aktiviert, da das Füllen von rechts nach links erfolgt und daher die einzelnen Words auch absteigend liegen müssen.

```
        move.l  #Pic,d0
        add.l   #101*40-2,d0
```

```
            move.l   d0,BLTAPTH(a6)
            move.l   d0,BLTDPTH(a6)
;
            move.w   #6400+20,BLTSIZE(a6)
            rts
```

Nach dem Eintrag der Quelle A und des Ziels D wird der Blitter durch Beschreiben von BLTSIZE gestartet.
In unserem Beispielprogramm wurden die Begrenzungslinien mit dem Linien-Modus des Blitters zuerst gezogen und anschließend der entstandene Zwischenraum aufgefüllt.

```
;
;           Flächen füllen
;
;
bltddat  =     $00
dmaconr  =     $02
bltcon0  =     $40
bltcon1  =     $42
bltafwm  =     $44
bltalwm  =     $46
bltcpth  =     $48
bltapth  =     $50
bltaptl  =     $52
bltdpth  =     $54
bltsize  =     $58
bltcmod  =     $60
bltbmod  =     $62
bltamod  =     $64
bltdmod  =     $66
bltcdat  =     $70
bltbdat  =     $72
bltadat  =     $74
;
s:          bsr.L    opening
```

```
          bsr.L     initcolors
;
          move.w    #150,d0              ; X1-Koordinate
          move.w    #0,d1                ; Y1-Koordinate
          move.w    #20,d2               ; X2-Koordinate
          move.w    #100,d3              ; Y2-Koordinate
          move.l    #Pic,a4              ; Adresse der Bitplane
          move.l    #$dff000,a6          ; Basisadresse
          move.w    #$ffff,bltbdat(a6)   ; Muster d. Linie
          bsr.L     Drawline             ; Routine anspringen
;
WaitLoop: btst      #6,DMACONR(A6)
          bne.s     WaitLoop
;
          move.w    #150,d0              ; X1-Koordinate
          move.w    #0,d1                ; Y1
          move.w    #300,d2              ; X2
          move.w    #100,d3              ; Y2
          move.l    #Pic,a4              ; Adresse der Bitplane
          move.l    #$dff000,a6          ; Basisadresse
          move.w    #$ffff,bltbdat(a6)   ; Muster d. Linie
          bsr.L     Drawline             ; Routine anspringen
;
WaitLoop2:btst      #6,DMACONR(a6)
          bne.s     WaitLoop2
;
          move.l    #$dff000,a6
          bsr.s     fill
;
loop:     btst      #6,$bfe001
          bne.s     loop
;
;
end:      move.w    #$c000,$dff09a
          moveq.l   #0,d0
e:        rts
```

```
opening:    move.w     #$4000,$dff09a
            move.l     #copperlist,$dff084
            move.l     #Pic,d0
            move       d0,pl1+6
            swap       d0
            move       d0,pl1+2
            swap       d0
;
            move.l     #sprite,d0
            lea        sp1+2(pc),a0
            moveq.l    #7,d1
initsprites: swap      d0
            move       d0,(a0)
            add.l      #4,a0
            swap       d0
            move       d0,(a0)
            add.l      #4,a0
            dbf        d1,initsprites
            rts
;
;
initcolors: lea        color(pc),a0
            lea        colortable(pc),a1
            moveq      #31,d0
            move       #$180,d1
copycolors:move        d1,(a0)+
            move       (a1)+,(a0)+
            addq       #2,d1
            dbf        d0,copycolors
            rts
;
;****************************************
;
fill:       move.w     #$FFFF,BLTDDAT(a6)
            move.w     #$FFFF,BLTCDAT(a6)
            move.w     #$FFFF,BLTBDAT(a6)
```

```
        move.w   #$FFFF,BLTADAT(a6)
;
        clr.w    BLTAMOD(a6)
        clr.w    BLTDMOD(a6)
        move.w   #$FFFF,BLTAFWM(a6)
        move.w   #$FFFF,BLTALWM(a6)
        move.w   #%0000000000001010,BLTCON1(a6)
        move.w   #%0000100111110000,BLTCON0(a6)
;
        move.l   #Pic,d0
        add.l    #101*40-2,d0
        move.l   d0,BLTAPTH(a6)
        move.l   d0,BLTDPTH(a6)
;
        move.w   #6400+20,BLTSIZE(a6)
        rts
;
;*****************************************************
;
DrawLine:movem.l D0-D5/A4,-(A7)
;
        sub.w    D0,D2
        bmi.s    x0gx1
        sub.w    D1,D3
        bmi.s    y0gy10
        cmp.w    D3,D2
        bmi.s    dygdx0
        moveq    #[4*4]!1,D4
        bra.s    Ok0
;
dygdx0: exg      D3,D2
        moveq    #1,D4
        bra.s    Ok0
;
y0gy10: neg.w    D3
        cmp.w    D3,D2
```

```
            bmi.s       dygdx1
            moveq       #[6*4]!1,D4
            bra.s       Ok0
;
dygdx1:     exg         D3,D2
            moveq       #[1*4]!1,D4
            bra.s       Ok0
;
x0gx1:      neg.w       D2
            sub.w       D1,D3
            bmi.s       y0gy11
            cmp.w       D3,D2
            bmi.s       dygdx2
            moveq       #[5*4]!1,D4
            bra.s       Ok0
;
dygdx2:     exg         D3,D2
            moveq       #[2*4]!1,D4
            bra.s       Ok0
;
y0gy11:     neg.w       D3
            cmp.w       D3,D2
            bmi.s       dygdx3
            moveq       #[7*4]!1,D4
            bra.s       Ok0
;
dygdx3:     exg         D3,D2
            moveq       #[3*4]!1,D4
;
;
Ok0:        lsl.w       #2,D3
;
WaitLoop3:btst          #6,DMACONR(A6)
            bne.s       WaitLoop3
;
            move.w      D3,BLTBMOD(A6)
```

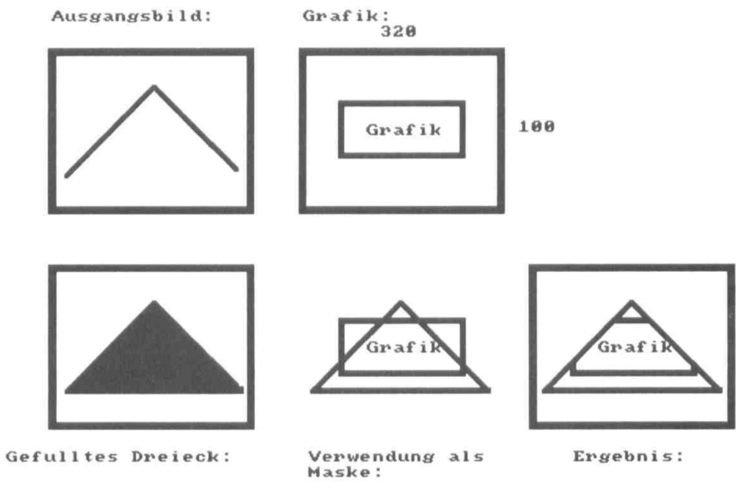
```
;
        add.w    D2,D2
        sub.w    D2,D3
        bpl.s    Ok1
        or.b     #%1000000,D4
Ok1:    move.w   D3,BLTAPTL(A6)
        or.w     #%1111000000000010,D4
        move.w   D4,BLTCON1(A6)
;
        sub.w    D2,D3
        move.w   D3,BLTAMOD(A6)
;
        move.w   #$8000,BLTADAT(A6)
        moveq    #-1,D3
        move.l   D3,BLTAFWM(A6)
        move.w   #40,BLTCMOD(A6)
        move.w   #40,BLTDMOD(A6)
;
        move.w   D0,D3
        and.w    #$f,D3
        ror.w    #4,D3
        or.w     #$bea,D3
        move.w   D3,BLTCON0(A6)
;
        lsr.w    #3,D0
        mulu     #40,D1
        add.w    D1,D0
        lea.l    (A4,D0.w),A4
        move.l   A4,BLTCPTH(A6)
        move.l   A4,BLTDPTH(A6)
;
        lsl.w    #5,D2
        add.w    #$42,D2
        move.w   D2,BLTSIZE(A6)
        movem.l  (A7)+,D0-D5/A4
        rts
```

```
;**********************************************
copperlist:
            dc.w      $008e,$3090
            dc.w      $0090,$f8f0
            dc.w      $0092,$0038
            dc.w      $0094,$00d0
            dc.w      $0102,$0000
            dc.w      $0104,$0024
            dc.w      $0108,$0000
            dc.w      $010a,$0000
            dc.w      $0100,$1200
pl1:        dc.w      $00e0,$0000
            dc.w      $00e2,$0000
color:      blk.w     64,0
sp1:        dc.w      $0120,$0000
            dc.w      $0122,$0000
            dc.w      $0124,$0000
            dc.w      $0126,$0000
            dc.w      $0128,$0000
            dc.w      $012a,$0000
            dc.w      $012c,$0000
            dc.w      $012e,$0000
            dc.w      $0130,$0000
            dc.w      $0132,$0000
            dc.w      $0134,$0000
            dc.w      $0136,$0000
            dc.w      $0138,$0000
            dc.w      $013a,$0000
            dc.w      $013c,$0000
            dc.w      $013e,$0000
            dc.w      $ffff,$fffe
;
;
colortable: dc.w      $0000,$0fff,$0fff,$0fff ;
sprite:     dc.       1 0
Pic:        blk.b     8000,0
```

3.5.2. FLÄCHEN MIT MUSTERN FÜLLEN

Wie man Flächen einfarbig füllt, haben wir bereits gehört. Nun wollen wir eine Fläche mit einem beliebigen Muster, also einer Grafik füllen. Für unser Beispiel reicht es, wenn diese Grafik mit der die Fläche ausgefüllt werden soll, nur eine Bitplane besitzt. Das Muster kann von beliebiger Gestalt sein. Sie können mit einem Malprogramm ein Bild erstellen, das 1 Bitplane (=2 Farben) besitzt und 320 mal 100 Punkte groß ist.

Der Blitter unterstützt eine solche Füll-Operation nicht, daher müssen wir zu einem Trick greifen. Die Fläche wird ganz normal ausgefüllt. Damit jetzt jedoch in der ausgefüllten Fläche das zuvor erstellte Bild erscheint, verwenden wir die gefüllte Fläche als Maske und "stanzen" somit quasi das gewünschte Bild aus.



Durch diesen Trick ist es möglich, ein beliebiges Muster in eine beliebigen Fläche zu plazieren.

```
;
;           Flächen mit Muster füllen
;
;
;           Nachzuladendes File:
;
;
;           "raw/Muster.raw"  auf  "muster"
;
bltddat     =           $00
dmaconr     =           $02
bltcon0     =           $40
bltcon1     =           $42
bltafwm     =           $44
bltalwm     =           $46
bltcpth     =           $48
bltbpth     =           $4c
bltapth     =           $50
bltaptl     =           $52
bltdpth     =           $54
bltsize     =           $58
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
s:          bsr.L       opening
            bsr.L       initcolors
;
            move.w      #150,d0             ; X1-Koordinate
            move.w      #0,d1               ; Y1
            move.w      #20,d2              ; X2
            move.w      #100,d3             ; Y2
            move.l      #Pic,a4             ; Adresse der Bitplane
            move.l      #$dff000,a6         ; Basisadresse
```

```
            move.w   #$ffff,bltbdat(a6)      ; Muster d. Linie
            bsr.L    Drawline                ; Routine anspringen
;
WaitLoop:   btst     #6,DMACONR(A6)
            bne.s    WaitLoop
;
            move.w   #150,d0                 ; X1-Koordinate
            move.w   #0,d1                   ; Y1
            move.w   #300,d2                 ; X2
            move.w   #100,d3                 ; Y2
            move.l   #Pic,a4                 ; Adresse der Bitplane
            move.l   #$dff000,a6             ; Basisadresse
            move.w   #$ffff,bltbdat(a6)      ; Muster d. Linie
            bsr.L    Drawline                ; Routine anspringen
;
WaitLoop2:btst       #6,DMACONR(a6)
            bne.s    WaitLoop2
;
            move.l   #$dff000,a6
            bsr.L    fill
;
loop:       btst     #6,$bfe001
            bne.s    loop
;
end:        move.w   #$c000,$dff09a
            moveq.l  #0,d0
e:          rts
;
opening:    move.w   #$4000,$dff09a
            move.l   #copperlist,$dff084
            move.l   #Pic,d0
            move     d0,pl1+6
            swap     d0
            move     d0,pl1+2
            swap     d0
;
```

```
            move.l     #sprite,d0
            lea        sp1+2(pc),a0
            moveq.l    #7,d1
initsprites: swap      d0
            move       d0,(a0)
            add.l      #4,a0
            swap       d0
            move       d0,(a0)
            add.l      #4,a0
            dbf        d1,initsprites
            rts
;
;
initcolors: lea        color(pc),a0
            lea        colortable(pc),a1
            moveq      #31,d0
            move       #$180,d1
copycolors:move        d1,(a0)+
            move       (a1)+,(a0)+
            addq       #2,d1
            dbf        d0,copycolors
;
            rts
;
;****************************************
;
fill:
;
; Füllen der Fläche
;
            move.w     #$FFFF,BLTDDAT(a6)
            move.w     #$FFFF,BLTCDAT(a6)
            move.w     #$FFFF,BLTBDAT(a6)
            move.w     #$FFFF,BLTADAT(a6)
;
            clr.w      BLTAMOD(a6)
```

```
        clr.w       BLTDMOD(a6)
        move.w      #$FFFF,BLTAFWM(a6)
        move.w      #$FFFF,BLTALWM(a6)
        move.w      #%0000000000001010,BLTCON1(a6)
        move.w      #%0000100111110000,BLTCON0(a6)
;
        move.l      #Pic,d0
        add.l       #101*40-2,d0
        move.l      d0,BLTAPTH(a6)
        move.l      d0,BLTDPTH(a6)
;
        move.w      #6400+20,BLTSIZE(a6)
;
WaitLoop3:btst      #6,DMACONR(a6)
        bne.s       WaitLoop3
;
; Gefüllte Fläche als Maske für das Muster verwenden
;
        clr.w       BLTAMOD(a6)
        clr.w       BLTBMOD(a6)
        clr.w       BLTDMOD(a6)
        move.w      #$FFFF,BLTAFWM(a6)
        move.w      #$FFFF,BLTALWM(a6)
        clr.w       BLTCON1(a6)
        move.w      #%0000110111000000,BLTCON0(a6)
;
        move.l      #Muster,BLTAPTH(a6)
        move.l      #Pic+40,BLTBPTH(a6)
        move.l      #Pic+40,BLTDPTH(a6)
;
        move.w      #6400+20,BLTSIZE(a6)
;
        rts
;
;****************************************************
;
```

```
DrawLine: movem.l   D0-D5/A4,-(A7)
;
          sub.w     0,D2
          bmi.s     x0gx1
          sub.w     D1,D3
          bmi.s     y0gy10
          cmp.w     D3,D2
          bmi.s     dygdx0
          moveq     #[4*4]!1,D4
          bra.s     Ok0
;
dygdx0:   exg       D3,D2
          moveq     #1,D4
          bra.s     Ok0
;
y0gy10:   neg.w     D3
          cmp.w     D3,D2
          bmi.s     dygdx1
          moveq     #[6*4]!1,D4
          bra.s     Ok0
;
dygdx1:   exg       D3,D2
          moveq     #[1*4]!1,D4
          bra.s     Ok0
;
x0gx1:    neg.w     D2
          sub.w     D1,D3
          bmi.s     y0gy11
          cmp.w     D3,D2
          bmi.s     dygdx2
          moveq     #[5*4]!1,D4
          bra.s     Ok0
;
dygdx2:   exg       D3,D2
          moveq     #[2*4]!1,D4
          bra.s     Ok0
```

```
;
y0gy11:  neg.w    D3
         cmp.w    D3,D2
         bmi.s    dygdx3
         moveq    #[7*4]!1,D4
         bra.s    Ok0
;
dygdx3:  exg      D3,D2
         moveq    #[3*4]!1,D4
;
;
Ok0:     lsl.w    #2,D3
;
WaitLoop4:
         btst     #6,DMACONR(A6)
         bne.s    WaitLoop4
;
         move.w   D3,BLTBMOD(A6)
;
         add.w    D2,D2
         sub.w    D2,D3
         bpl.s    Ok1
         or.b     #%1000000,D4
Ok1:     move.w   D3,BLTAPTL(A6)
         or.w     #%1111000000000010,D4
         move.w   D4,BLTCON1(A6)
;
         sub.w    D2,D3
         move.w   D3,BLTAMOD(A6)
;
         move.w   #$8000,BLTADAT(A6)
         moveq    #-1,D3
         move.l   D3,BLTAFWM(A6)
         move.w   #40,BLTCMOD(A6)
         move.w   #40,BLTDMOD(A6)
;
```

```
            move.w    D0,D3
            and.w     #$f,D3
            ror.w     #4,D3
            or.w      #$bea,D3
            move.w    D3,BLTCON0(A6)
;
            lsr.w     #3,D0
            mulu      #40,D1
            add.w     D1,D0
            lea.l     (A4,D0.w),A4
            move.l    A4,BLTCPTH(A6)
            move.l    A4,BLTDPTH(A6)
;
            lsl.w     #5,D2
            add.w     #$42,D2
            move.w    D2,BLTSIZE(A6)
            movem.l   (A7)+,D0-D5/A4
            rts
;
;***********************************************
;
copperlist:
            dc.w      $008e,$3090
            dc.w      $0090,$f8f0
            dc.w      $0092,$0038
            dc.w      $0094,$00d0
            dc.w      $0102,$0000
            dc.w      $0104,$0024
            dc.w      $0108,$0000
            dc.w      $010a,$0000
            dc.w      $0100,$1200
pl1:        dc.w      $00e0,$0000
            dc.w      $00e2,$0000
sp1:        dc.w      $0120,$0000
            dc.w      $0122,$0000
            dc.w      $0124,$0000
```

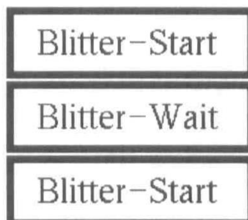
```
            dc.w        $0126,$0000
            dc.w        $0128,$0000
            dc.w        $012a,$0000
            dc.w        $012c,$0000
            dc.w        $012e,$0000
            dc.w        $0130,$0000
            dc.w        $0132,$0000
            dc.w        $0134,$0000
            dc.w        $0136,$0000
            dc.w        $0138,$0000
            dc.w        $013a,$0000
            dc.w        $013c,$0000
            dc.w        $013e,$0000
color:      blk.w       64,0
            dc.w        $ffff,$fffe
colortable: dc.w        $0000,$0fff,$0fff,$0fff
;
;
sprite:     dc.         1 0
Pic:        blk.b       8000,0
Muster:     blk.b       4000,0
;
```

3.6. TIPS & TRICKS ZUM BLITTER

Um das in den vorhergehenden Kapiteln erworbene Wissen zu vertiefen, aber auch um neue Anwendungsgebiete kennenzulernen, sind hier einige Beispiele zur Blitter-Programmierung aufgeführt.

3.6.1. SINNVOLLE ÜBERBRÜCKUNG DER BLITTERWAITS

Aus den vorherigen Kapiteln wissen wir bereits, daß man zwischen zwei Blitteroperationen immer warten muß, bis der Blitter seine Aufgabe beendet hat, bevor man ihn erneut starten kann.
Hat man aber nun mehrere Aufgaben für den Blitter, so können mitunter längere Wartezeiten entstehen, in denen der Prozessor eigentlich nichts anderes tut, als auf das Signal des Blitters zu warten, damit er neu initialisiert werden kann.
Sehen wir dazu die Skizze.



Ist der Bereich, den der Blitter kopieren soll, etwas größer, so entsteht eine ziemlich lange Wartezeit für den Prozessor, der in der Warteschleife verharrt. Diese Zeit kann man aber auch sinnvoll verwenden, indem man in dieser Zeitspanne anfallende Berechnungen vom Prozessor erledigen läßt. Dadurch wird ebenfalls erheblich Zeit gespart, denn wenn die Berechnungen gleichzeitig mit dem Blitter erledigt werden, kann man nach Beendigung der Blitteraktion schon die fertigen Werte der Berechnungen für die nächste Blitteroperation verwenden (Siehe Abbildung uf der nächsten Seite).



Werden sehr große Bereiche mit dem Blitter kopiert, wie zum Beispiel ein großes Bob, sodaß der Blit-Vorgang sehr viel Zeit in Anspruch nimmt, kann man während dieser Zeit auch aufwendigere Routinen, die unterschiedlich viel Zeit in Anspruch nehmen, anspringen. Dies wäre etwa die Abfrage für den Joystick, wobei je nach Hebelstellung ein Sprite über den Bildschirm bewegt wird.

3.6.2. 5 PLANES MIT EINEM BLIT

Im Kapitel "Kopieren" haben wir den ersten Umgang mit dem Blitter kennengelernt und wissen jetzt, wie man ein Objekt auf den Bildschirm bringt. Besteht dieses Bob aber aus mehr als einer Bitplane, so haben wir in unseren Beispielprogrammen jede Bitplane einzeln kopiert. Also für jede Bitplane den Blitter erneut gestartet. Das kostet natürlich viel Zeit. Besonders bei den folgenden Listings wird es notwendig sein, Rasterzeit einzusparen. Daher bedienen wir uns hier eines Tricks, um gleich alle 5 Bitplanes mit nur einem Blitterstart kopieren zu können.

Dazu sind aber einige nicht unwesentliche Vorbereitungen notwendig. Wir erinnern uns, die ims RAW-Format konvertierten Bitplanes liegen nacheinander im Speicher. Genauer gesagt:

1. Zeile der 1. Bitplane
2. Zeile der 1. Bitplane
.
.
letzte Zeile der 1. Bitplane

1. Zeile der 2. Bitplane
2. Zeile der 2. Bitplane
.
.
letzte Zeile der 2. Bitplane
u.s.w.

Liegen die Bitplanes, wie oben gezeigt im Speicher, so bleibt uns nichts anderes übrig, als jede Bitplane einzeln zu kopieren. Wenn wir die Bitplanes jedoch anders ordnen, können wir sie mit nur einem Blitterstart kopieren. Nämlich:

1. Zeile der 1. Bitplane
1. Zeile der 2. Bitplane
1. Zeile der 3. Bitplane

2. Zeile der 1. Bitplane
2. Zeile der 2. Bitplane
2. Zeile der 3. Bitplane

.
.

letzte Zeile der 1. Bitplane
letzte Zeile der 2. Bitplane
letzte Zeile der 3. Bitplane

Sind die Bitplanes so geordnet, kann man sie mit nur einem Blit so verschieben, daß man, egal ob man 2 oder gar 6 Bitplanes verwendet, nur einen Blitterstart benötigt.

Doch wie bringt man die Zeilen eines ins RAW-Format konvertierten Bildes nun in die gewünschte Reihenfolge?

Entweder schreiben Sie ein eigenes Programm, das dies erledigt, oder Sie verwenden das dem Buch beigelegte Programm "Bitplane-Mixer". Dieser Bitplane-Mixer übernimmt die Arbeit des "Bitplane-Umschichtens" für Sie. Eine ausführliche Bedienungsanleitung finden Sie im Anhang.

Wie man die fertig umgeschlichteten Bildplanes mit nur einem Blitterstart kopiert, zeigt das abschließende Listing:

```
;
;           5 Planes mit 1 Blit
;
;           Nachzuladendes File:
;
;           "raw/Bob32x32_5.con" auf "bob"
;
dmaconr  =        $02
bltddat  =        $00
bltcon0  =        $40
bltcon1  =        $42
bltafwm  =        $44
bltalwm  =        $46
bltcpth  =        $48
bltbpth  =        $4c
bltapth  =        $50
bltaptl  =        $52
bltdpth  =        $54
bltsize  =        $58
bltcmod  =        $60
bltbmod  =        $62
bltamod  =        $64
bltdmod  =        $66
bltcdat  =        $70
```

```
bltbdat     =         $72
bltadat     =         $74
;
s:          bsr.s     opening
            bsr.L     initcolors
;
;
            move.l    #$DFF000,a6
            bsr.L     Blit                  ;Routine anspringen
;
loop:       btst      #6,$bfe001
            bne.s     loop
;
;
end:        move.w    #$c000,$dff09a
            moveq.l   #0,d0
e:          rts
;
opening:    move.w    #$4000,$dff09a
            move.l    #copperl,$dff084
            move.l    #pic,d0
            move      d0,pl1+6
            swap      d0
            move      d0,pl1+2
            swap      d0
            add.l     size(pc),d0
            move      d0,pl2+6
            swap      d0
            move      d0,pl2+2
            swap      d0
            add.l     size(pc),d0
            move      d0,pl3+6
            swap      d0
            move      d0,pl3+2
            swap      d0
            add.l     size(pc),d0
```

```
            move        d0,pl4+6
            swap        d0
            move        d0,pl4+2
            swap        d0
            add.l       size(pc),d0
            move        d0,pl5+6
            swap        d0
            move        d0,pl5+2
            swap        d0
;
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.l     #7,d1
initsprites: swap       d0
            move        d0,(a0)
            add.l       #4,a0
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
;
initcolors: lea         color(pc),a0
            lea         bob+640(pc),a1
            moveq       #31,d0
            move        #$180,d1
copycolors:move         d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
;
            rts
;
copperl:
            dc.w        $008e,$3090
```

```
                dc.w      $0090,$f8f0
                dc.w      $0092,$0038
                dc.w      $0094,$00d0
                dc.w      $0102,$0000
                dc.w      $0104,$0024
                dc.w      $0108,$00a0
                dc.w      $010a,$00a0
                dc.w      $0100,$5300
pl1:            dc.w      $00e0,$0000
                dc.w      $00e2,$0000
pl2:            dc.w      $00e4,$0000
                dc.w      $00e6,$0000
pl3:            dc.w      $00e8,$0000
                dc.w      $00ea,$0000
pl4:            dc.w      $00ec,$0000
                dc.w      $00ee,$0000
pl5:            dc.w      $00f0,$0000
                dc.w      $00f2,$0000
sp1:            dc.w      $0120,$0000
                dc.w      $0122,$0000
                dc.w      $0124,$0000
                dc.w      $0126,$0000
                dc.w      $0128,$0000
                dc.w      $012a,$0000
                dc.w      $012c,$0000
                dc.w      $012e,$0000
                dc.w      $0130,$0000
                dc.w      $0132,$0000
                dc.w      $0134,$0000
                dc.w      $0136,$0000
                dc.w      $0138,$0000
                dc.w      $013a,$0000
                dc.w      $013c,$0000
                dc.w      $013e,$0000
color:          blk.w     64,0
                dc.w      $ffff,$fffe
```

```
;
size:       dc.         l 40
;
;*****************************************
;
Blit:
            move.w      #%0000100111110000,BLTCON0(A6)
            clr.w       BLTCON1(a6)
;
            clr.w       BLTAMOD(a6)
            move.w      #36,BLTDMOD(a6)
;
            move.w      #$FFFF,BLTAFWM(a6)
            move.w      #$FFFF,BLTALWM(a6)
;
            move.l      #bob,BLTAPTH(a6)  ;Adresse d. Bobs
;
            move.l      #pic,d0           ;Adresse d. Bitplane
            add.l       #12816,d0         ;Position
            move.l      d0,BLTDPTH(a6)
;
            move.w      #10242,BLTSIZE(a6)
;
            rts
;
;*****************************************
;
sprite:     dc.l        0
bob:        blk.b       704,0
pic:        blk.b       40000,0
;
```

3.6.3. 5 PLANES MIT MASKE MIT EINEM BLIT

Man kann sogar noch mehr Zeit sparen, wenn man nicht nur alle 5 Bitplanes eines Objektes mit einem Blitterstart kopiert, sondern falls notwendig, auch gleich die Maske mitkopiert. Die Bitplanes der Maske müssen natürlich auch so angeordnet sein, daß man Sie mit einem "Blit" im Speicher verschieben kann. Das heißt, sie müssen ebenfalls mit dem Bitplane-Mixer nebeneinander gelegt werden.

Das untenstehende Beispielprogramm zeigt, wie man ein 32 mal 32 großes Bob mit 5 Bitplanes und einer dazugehörigen Maske (ebenfalls mit 5 Bitplanes) in ein 32 farbiges Hintergrundbild kopiert, ohne die hinter dem Bob liegende Grafik zu zerstören.
Dieses Kopierverfahren wird in den nachfolgenden Listings (vor allem bei den Laufschriften) angewandt. Deshalb müssen Sie sämtliche Grafiken, die aus mehreren Planes bestehen, mit dem Bitplane-Mixer bearbeiten, damit die Bitplanes nebeneinander zu liegen kommen.

```
;
;		5 Planes mit Maske mit einem Blit
;
;		Nachzuladende Files:
;
;		"raw/Hintergrund.con" auf "pic"
;		"raw/Bob32x32_5.con" auf "bob"
;		"raw/Bob32x32_5_msk.con" auf "msk"
;
dmaconr	=	$02
bltddat	=	$00
bltcon0	=	$40
bltcon1	=	$42
bltafwm	=	$44
bltalwm	=	$46
bltcpth	=	$48
bltbpth	=	$4c
```

```
bltapth     =          $50
bltaptl     =          $52
bltdpth     =          $54
bltsize     =          $58
bltcmod     =          $60
bltbmod     =          $62
bltamod     =          $64
bltdmod     =          $66
bltcdat     =          $70
bltbdat     =          $72
bltadat     =          $74
;
s:          bsr.s      opening
            bsr.L      initcolors
;
;
            move.l     #$DFF000,a6
            bsr.L      Blit                   ;Routine anspringen
;
loop:       btst       #6,$bfe001
            bne.s      loop
;
;
end:        move.w     #$c000,$dff09a
            moveq.l    #0,d0
e:          rts
;
opening:    move.w     #$4000,$dff09a
            move.l     #copperl,$dff084
            move.l     #pic,d0
            move       d0,pl1+6
            swap       d0
            move       d0,pl1+2
            swap       d0
            add.l      size(pc),d0
            move       d0,pl2+6
```

```
            swap      d0
            move      d0,pl2+2
            swap      d0
            add.l     size(pc),d0
            move      d0,pl3+6
            swap      d0
            move      d0,pl3+2
            swap      d0
            add.l     size(pc),d0
            move      d0,pl4+6
            swap      d0
            move      d0,pl4+2
            swap      d0
            add.l     size(pc),d0
            move      d0,pl5+6
            swap      d0
            move      d0,pl5+2
            swap      d0
;
            move.l    #sprite,d0
            lea       sp1+2(pc),a0
            moveq.l   #7,d1
initsprites: swap     d0
            move      d0,(a0)
            add.l     #4,a0
            swap      d0
            move      d0,(a0)
            add.l     #4,a0
            dbf       d1,initsprites
            rts
;
;
initcolors: lea       color(pc),a0
            lea       pic+40000,a1
            moveq     #31,d0
            move      #$180,d1
```

```
copycolors:move     d1,(a0)+
           move     (a1)+,(a0)+
           addq     #2,d1
           dbf      d0,copycolors
;
           rts
;
copperl:
           dc.w     $008e,$3090
           dc.w     $0090,$f8f0
           dc.w     $0092,$0038
           dc.w     $0094,$00d0
           dc.w     $0102,$0000
           dc.w     $0104,$0024
           dc.w     $0108,$00a0
           dc.w     $010a,$00a0
           dc.w     $0100,$5300
pl1:       dc.w     $00e0,$0000
           dc.w     $00e2,$0000
pl2:       dc.w     $00e4,$0000
           dc.w     $00e6,$0000
pl3:       dc.w     $00e8,$0000
           dc.w     $00ea,$0000
pl4:       dc.w     $00ec,$0000
           dc.w     $00ee,$0000
pl5:       dc.w     $00f0,$0000
           dc.w     $00f2,$0000
sp1:       dc.w     $0120,$0000
           dc.w     $0122,$0000
           dc.w     $0124,$0000
           dc.w     $0126,$0000
           dc.w     $0128,$0000
           dc.w     $012a,$0000
           dc.w     $012c,$0000
           dc.w     $012e,$0000
           dc.w     $0130,$0000
```

```
        dc.w    $0132,$0000
        dc.w    $0134,$0000
        dc.w    $0136,$0000
        dc.w    $0138,$0000
        dc.w    $013a,$0000
        dc.w    $013c,$0000
        dc.w    $013e,$0000
color:  blk.w   64,0
        dc.w    $ffff,$fffe
;
size:   dc.     l 40
;
;*****************************************
;
Blit:
        move.w  #%0000111111000010,BLTCON0(A6)
        clr.w   BLTCON1(a6)
;
        clr.w   BLTAMOD(a6)
        clr.w   BLTBMOD(a6)
        move.w  #36,BLTDMOD(a6)
        move.w  #36,BLTCMOD(a6)
;
        move.w  #$FFFF,BLTAFWM(a6)
        move.w  #$FFFF,BLTALWM(a6)
;
        move.l  #bob,BLTAPTH(a6)  ;Adresse d. Bobs
;
        move.l  #msk,BLTBPTH(a6)  ;Adresse d.
                                  ;Bobmaske
;
        move.l  #pic,d0           ;Adresse d. Bitplane
        add.l   #12816,d0         ;Position
        move.l  d0,BLTDPTH(a6)
        move.l  d0,BLTCPTH(a6)
;
```

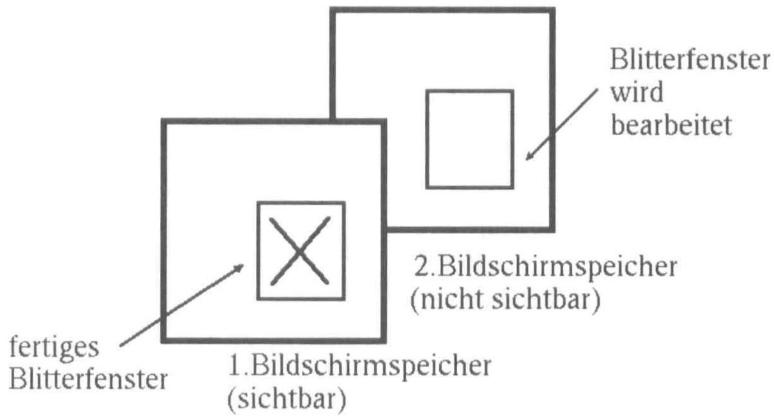
```
         move.w    #10242,BLTSIZE(a6)
;
         rts
;
;*******************************************
;
sprite:  dc.l      0
bob:     blk.b     704,0
msk:     blk.b     704,0
pic:     blk.b     40064,0
;
```

3.6.4. DOUBLEBUFFERING

Wenn trotz aller Tricks und Optimierungen die Berechnungen eines Programmes zu lange dauern, sodaß sie sich mit der Bilddarstellung überschneiden würden, muß man auf eine andere Methode zurückgreifen, die es ermöglicht, das Bild trotzdem ruckelfrei darzustellen. Diese Technik nennt man Doublebuffering.

Die Erklärung ist ziemlich einfach. Normalerweise verwendet man einen Bildschirmspeicher und ändert die Bitplanes bevor diese vom Rasterstrahl neu aufgebaut werden. Da dies aus zeitlichen Gründen bei längeren Berechnungen nicht mehr möglich ist, ohne daß das Bild flackert, verwendet man einfach einen zweiten, nicht sichtbaren Bildschirmspeicher (siehe Abbildung auf der nächsten Seite).

Auf dem nicht sichtbaren Bildschirmspeicher kann man ungestört das Bild bearbeiten und aufbauen, da es ja vom Betrachter nicht wahrgenommen wird. Ist der Aufbau beendet, tauscht man einfach die beiden Bildschirmspeicher aus, sodaß der jetzt fertig aufgebaute im Vordergrund ist, also jetzt sichtbar ist. Der andere hingegen ist nicht sichtbar und wird neu aufgebaut.



Durch diese Technik kann man auch komplizierte Bildschirmaufbauten gestalten, ohne daß das fertige Bild in irgendeiner Form beeinträchtigt wird.

4. DAS AGA-CHIPSET

Obwohl die Unterschiede zum älteren ECS Chipset ziemliche Fortschritte, z.B.: in der Farbdarstellung bringen, ist die Programmierung nicht allzu schwer. Es gibt lediglich einige neue Register, deren Funktion man kennen sollte, sowie einige Richtlinien, die man befolgen sollte, um die Kompatibilität zu wahren.

ALLGEMEINE PROBLEME

Leider ist auch das neue Chipset nicht ganz fehlerfrei. Ein wichtiger Hinweis, den Sie unbedingt beachten sollten, liegt gleich am Beginn der Copperliste versteckt: Wenn Sie eine Copperliste für AGA-Rechner schreiben, sollten Sie am Anfang auf jeden Fall mittels WAIT-Befehl auf Zeile 2 warten.

```
Coplist:
        dc.w    $0207,$fffe     ;You have to wait at
                                ; least for line 2 to
                                ; avoid troubles.
```

Tut man das nicht, kann es zu Problemen kommen.

DEN RICHTIGEN SCREENMODUS ERKENNEN UND UMSCHALTEN

Viele Spiele haben Probleme, wenn man bei den neuen Amigas eine andere Bildschirmauflösung als 320 x 200 oder 640 x 480 eingestellt hat. Besonders, wenn der Double Scan Modus aktiviert ist, läßt sich das eigene Programm nicht mehr starten.
Um das Problem zu umgehen, sollte man am Anfang der eigenen Copperliste folgende Zeilen einfügen:

```
dc.w    $01c0,$0000             ;Clear all AGA special
                                ; infos to avoid troubles
dc.w    $01c2,$0000             ; with double scan
dc.w    $01c4,$0000             ; modes etc.
dc.w    $01c6,$0000             ; Save old values out of
dc.w    $01c8,$0000             ; read registers to
                                ; restore after finishing
dc.w    $01ca,$0000             ; (if neccessary)
dc.w    $01cc,$0000
dc.w    $01ce,$0000
dc.w    $01d0,$0000
dc.w    $01d2,$0000
dc.w    $01d4,$0000
dc.w    $01d6,$0000
dc.w    $01d8,$0000
dc.w    $01da,$0000
dc.w    $01dc,%0000000000100000     ;PAL
dc.w    $01de,$0000

dc.w    $01e0,$0000
dc.w    $01e2,$0000
dc.w    $01e4,$0000
dc.w    $01e6,$0000
dc.w    $01e8,$0000
dc.w    $01ea,$0000
dc.w    $01ec,$0000
dc.w    $01ee,$0000
```

Hier werden sämtliche Werte zurückgesetzt, damit ein Umschalten auf die eigene Bildschirmauflösung gegeben ist.

INITIALISIEREN DES BILDSCHIRMMODUS, DER BIT-PLANES UND DER SPRITES

Nun werden, wie schon in Band 1 ausführlich besprochen, die eigenen Werte für die Bildschirmauflösung in die dafür vorgesehenen Register eingetragen. Wie sich diese Werte berechnen lassen, können Sie in Band 1 nachlesen, wir gehen deshalb hier nicht näher darauf ein.

```
        dc.w    $01fc,$0000             ;Fetchmode off
        dc.w    $0100,%0000000000010000     ;8 pitmaps,
        dc.w    $0102,$0000             ;No double playfield
        dc.w    $0104,%100100           ;No double pf. priority
        ;       $0106                   ;later at colorbanks
        dc.w    $010c,$0000             ;No spritecols etc.
        dc.w    $0108,$0000             ;modulo 0
        dc.w    $010a,$0000             ;modulo 0

        dc.w    $008e,$2c81             ;Displaywindow start
        dc.w    $0090,$f4c1             ;Displaywindow stop
        dc.w    $0092,$0038             ;Datafetch start
        dc.w    $0094,$00D0             ;Datafetch stop
```

Das Setzen der Bitplanes ist ebenfalls unverändert zur herkömmlichen Copperliste:

```
* 8 Pitplanes Pointer *
Bitmaps: dc.w   $00e0,0
         dc.w   $00e2,0
         dc.w   $00e4,0
         dc.w   $00e6,0
         dc.w   $00e8,0
         dc.w   $00ea,0
         dc.w   $00ec,0
         dc.w   $00ee,0
         dc.w   $00f0,0
```

```
        dc.w    $00f2,0
        dc.w    $00f4,0
        dc.w    $00f6,0
        dc.w    $00f8,0
        dc.w    $00fa,0
        dc.w    $00fc,0
        dc.w    $00fe,0
```

Auch beim Initialisieren der Sprites hat sich nichts geändert:

```
Sprite: dc.w    $0120,$0000
        dc.w    $0122,$0000
        dc.w    $0124,$0000
        dc.w    $0126,$0000
        dc.w    $0128,$0000
        dc.w    $012a,$0000
        dc.w    $012c,$0000
        dc.w    $012e,$0000
        dc.w    $0130,$0000
        dc.w    $0132,$0000
        dc.w    $0134,$0000
        dc.w    $0136,$0000
        dc.w    $0138,$0000
        dc.w    $013a,$0000
        dc.w    $013c,$0000
        dc.w    $013e,$0000
```

FARBEN SETZEN IM NEUEN AGA CHIPSET

Hier unterscheidet sich das neue vom alten Chipset gewaltig. Denn obwohl man jetzt 256 Farben gleichzeitig darstellen kann, gibt es nicht mehr Farbregister. Das Setzen der Farben an sich geschieht wie bisher. Jedoch ist beim AGA-Chipset ein neues Register hinzugekommen, Register $dff106, das bestimmt, auf welche „Farbbank" zugegriffen werden soll.

Insgesamt gibt es acht Farbbanken in denen die Farben eingetragen werden müssen. Mit Register $dff106 bestimmt man die Bank. In unserem Beispiel wurden alle Farben auf Null (=Farbe Schwarz) gesetzt).

```
        dc.w    $0106,%0000000001000000     ;Bank 0
cols:   dc.w    $0180,$0000             ;1st 32 colors, MSB
        dc.w    $0182,$0000
        dc.w    $0184,$0000
        dc.w    $0186,$0000
        dc.w    $0188,$0000
        dc.w    $018a,$0000
        dc.w    $018c,$0000
        dc.w    $018e,$0000
        dc.w    $0190,$0000
        dc.w    $0192,$0000
        dc.w    $0194,$0000
        dc.w    $0196,$0000
        dc.w    $0198,$0000
        dc.w    $019a,$0000
        dc.w    $019c,$0000
        dc.w    $019e,$0000
        dc.w    $01a0,$0000
        dc.w    $01a2,$0000
        dc.w    $01a4,$0000
        dc.w    $01a6,$0000
        dc.w    $01a8,$0000
        dc.w    $01aa,$0000
        dc.w    $01ac,$0000
        dc.w    $01ae,$0000
        dc.w    $01b0,$0000
        dc.w    $01b2,$0000
        dc.w    $01b4,$0000
        dc.w    $01b6,$0000
        dc.w    $01b8,$0000
        dc.w    $01ba,$0000
```

```
dc.w	$01bc,$0000
dc.w	$01be,$0fff

dc.w	$0106,%0000001001000000		;Bank 0
dc.w	$0180,$0000		;2nd 32 colors, LSB
dc.w	$0182,$0000
dc.w	$0184,$0000
dc.w	$0186,$0000
dc.w	$0188,$0000
dc.w	$018a,$0000
dc.w	$018c,$0000
dc.w	$018e,$0000
dc.w	$0190,$0000
dc.w	$0192,$0000
dc.w	$0194,$0000
dc.w	$0196,$0000
dc.w	$0198,$0000
dc.w	$019a,$0000
dc.w	$019c,$0000
dc.w	$019e,$0000
dc.w	$01a0,$0000
dc.w	$01a2,$0000
dc.w	$01a4,$0000
dc.w	$01a6,$0000
dc.w	$01a8,$0000
dc.w	$01aa,$0000
dc.w	$01ac,$0000
dc.w	$01ae,$0000
dc.w	$01b0,$0000
dc.w	$01b2,$0000
dc.w	$01b4,$0000
dc.w	$01b6,$0000
dc.w	$01b8,$0000
dc.w	$01ba,$0000
dc.w	$01bc,$0000
dc.w	$01be,$0000
```

```
dc.w    $0106,%0010000001000000     ;Bank 1
dc.w    $0180,$0000                 ;1st 32 colors, MSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w	$0106,%0010001001000000	;Bank 1
dc.w	$0180,$0000		;2nd 32 colors, LSB
dc.w	$0182,$0000
dc.w	$0184,$0000
dc.w	$0186,$0000
dc.w	$0188,$0000
dc.w	$018a,$0000
dc.w	$018c,$0000
dc.w	$018e,$0000
dc.w	$0190,$0000
dc.w	$0192,$0000
dc.w	$0194,$0000
dc.w	$0196,$0000
dc.w	$0198,$0000
dc.w	$019a,$0000
dc.w	$019c,$0000
dc.w	$019e,$0000
dc.w	$01a0,$0000
dc.w	$01a2,$0000
dc.w	$01a4,$0000
dc.w	$01a6,$0000
dc.w	$01a8,$0000
dc.w	$01aa,$0000
dc.w	$01ac,$0000
dc.w	$01ae,$0000
dc.w	$01b0,$0000
dc.w	$01b2,$0000
dc.w	$01b4,$0000
dc.w	$01b6,$0000
dc.w	$01b8,$0000
dc.w	$01ba,$0000
dc.w	$01bc,$0000
dc.w	$01be,$0000
```

```
	dc.w	$0106,%0100000001000000		;Bank 2
	dc.w	$0180,$0000			;1st 32 colors, MSB
	dc.w	$0182,$0000
	dc.w	$0184,$0000
	dc.w	$0186,$0000
	dc.w	$0188,$0000
	dc.w	$018a,$0000
	dc.w	$018c,$0000
	dc.w	$018e,$0000
	dc.w	$0190,$0000
	dc.w	$0192,$0000
	dc.w	$0194,$0000
	dc.w	$0196,$0000
	dc.w	$0198,$0000
	dc.w	$019a,$0000
	dc.w	$019c,$0000
	dc.w	$019e,$0000
	dc.w	$01a0,$0000
	dc.w	$01a2,$0000
	dc.w	$01a4,$0000
	dc.w	$01a6,$0000
	dc.w	$01a8,$0000
	dc.w	$01aa,$0000
	dc.w	$01ac,$0000
	dc.w	$01ae,$0000
	dc.w	$01b0,$0000
	dc.w	$01b2,$0000
	dc.w	$01b4,$0000
	dc.w	$01b6,$0000
	dc.w	$01b8,$0000
	dc.w	$01ba,$0000
	dc.w	$01bc,$0000
	dc.w	$01be,$0000
```

```
dc.w    $0106,%0100001001000000     ;Bank 2
dc.w    $0180,$0000                 ;2nd 32 colors, LSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w     $0106,%0110000001000000     ;Bank 3
dc.w     $0180,$0000                 ;1st 32 colors, MSB
dc.w     $0182,$0000
dc.w     $0184,$0000
dc.w     $0186,$0000
dc.w     $0188,$0000
dc.w     $018a,$0000
dc.w     $018c,$0000
dc.w     $018e,$0000
dc.w     $0190,$0000
dc.w     $0192,$0000
dc.w     $0194,$0000
dc.w     $0196,$0000
dc.w     $0198,$0000
dc.w     $019a,$0000
dc.w     $019c,$0000
dc.w     $019e,$0000
dc.w     $01a0,$0000
dc.w     $01a2,$0000
dc.w     $01a4,$0000
dc.w     $01a6,$0000
dc.w     $01a8,$0000
dc.w     $01aa,$0000
dc.w     $01ac,$0000
dc.w     $01ae,$0000
dc.w     $01b0,$0000
dc.w     $01b2,$0000
dc.w     $01b4,$0000
dc.w     $01b6,$0000
dc.w     $01b8,$0000
dc.w     $01ba,$0000
dc.w     $01bc,$0000
dc.w     $01be,$0000
```

```
dc.w	$0106,%0110001001000000	;Bank 3
dc.w	$0180,$0000		;2nd 32 colors, LSB
dc.w	$0182,$0000
dc.w	$0184,$0000
dc.w	$0186,$0000
dc.w	$0188,$0000
dc.w	$018a,$0000
dc.w	$018c,$0000
dc.w	$018e,$0000
dc.w	$0190,$0000
dc.w	$0192,$0000
dc.w	$0194,$0000
dc.w	$0196,$0000
dc.w	$0198,$0000
dc.w	$019a,$0000
dc.w	$019c,$0000
dc.w	$019e,$0000
dc.w	$01a0,$0000
dc.w	$01a2,$0000
dc.w	$01a4,$0000
dc.w	$01a6,$0000
dc.w	$01a8,$0000
dc.w	$01aa,$0000
dc.w	$01ac,$0000
dc.w	$01ae,$0000
dc.w	$01b0,$0000
dc.w	$01b2,$0000
dc.w	$01b4,$0000
dc.w	$01b6,$0000
dc.w	$01b8,$0000
dc.w	$01ba,$0000
dc.w	$01bc,$0000
dc.w	$01be,$0000
```

```
dc.w	$0106,%1000000001000000	;Bank 4
dc.w	$0180,$0000		;1st 32 colors, MSB
dc.w	$0182,$0000
dc.w	$0184,$0000
dc.w	$0186,$0000
dc.w	$0188,$0000
dc.w	$018a,$0000
dc.w	$018c,$0000
dc.w	$018e,$0000
dc.w	$0190,$0000
dc.w	$0192,$0000
dc.w	$0194,$0000
dc.w	$0196,$0000
dc.w	$0198,$0000
dc.w	$019a,$0000
dc.w	$019c,$0000
dc.w	$019e,$0000
dc.w	$01a0,$0000
dc.w	$01a2,$0000
dc.w	$01a4,$0000
dc.w	$01a6,$0000
dc.w	$01a8,$0000
dc.w	$01aa,$0000
dc.w	$01ac,$0000
dc.w	$01ae,$0000
dc.w	$01b0,$0000
dc.w	$01b2,$0000
dc.w	$01b4,$0000
dc.w	$01b6,$0000
dc.w	$01b8,$0000
dc.w	$01ba,$0000
dc.w	$01bc,$0000
dc.w	$01be,$0000
```

```
dc.w    $0106,%1000001001000000    ;Bank 4
dc.w    $0180,$0000                ;2nd 32 colors, LSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w    $0106,%1010000001000000   ;Bank 5
dc.w    $0180,$0000               ;1st 32 colors, MSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w    $0106,%1010001001000000    ;Bank 5
dc.w    $0180,$0000                ;2nd 32 colors, LSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w     $0106,%1100000001000000     ;Bank 6
dc.w     $0180,$0000                 ;1st 32 colors, MSB
dc.w     $0182,$0000
dc.w     $0184,$0000
dc.w     $0186,$0000
dc.w     $0188,$0000
dc.w     $018a,$0000
dc.w     $018c,$0000
dc.w     $018e,$0000
dc.w     $0190,$0000
dc.w     $0192,$0000
dc.w     $0194,$0000
dc.w     $0196,$0000
dc.w     $0198,$0000
dc.w     $019a,$0000
dc.w     $019c,$0000
dc.w     $019e,$0000
dc.w     $01a0,$0000
dc.w     $01a2,$0000
dc.w     $01a4,$0000
dc.w     $01a6,$0000
dc.w     $01a8,$0000
dc.w     $01aa,$0000
dc.w     $01ac,$0000
dc.w     $01ae,$0000
dc.w     $01b0,$0000
dc.w     $01b2,$0000
dc.w     $01b4,$0000
dc.w     $01b6,$0000
dc.w     $01b8,$0000
dc.w     $01ba,$0000
dc.w     $01bc,$0000
dc.w     $01be,$0000
```

```
dc.w    $0106,%1100001001000000    ;Bank 6
dc.w    $0180,$0000                ;2nd 32 colors, LSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w    $0106,%1110000001000000    ;Bank 7
dc.w    $0180,$0000                ;1st 32 colors, MSB
dc.w    $0182,$0000
dc.w    $0184,$0000
dc.w    $0186,$0000
dc.w    $0188,$0000
dc.w    $018a,$0000
dc.w    $018c,$0000
dc.w    $018e,$0000
dc.w    $0190,$0000
dc.w    $0192,$0000
dc.w    $0194,$0000
dc.w    $0196,$0000
dc.w    $0198,$0000
dc.w    $019a,$0000
dc.w    $019c,$0000
dc.w    $019e,$0000
dc.w    $01a0,$0000
dc.w    $01a2,$0000
dc.w    $01a4,$0000
dc.w    $01a6,$0000
dc.w    $01a8,$0000
dc.w    $01aa,$0000
dc.w    $01ac,$0000
dc.w    $01ae,$0000
dc.w    $01b0,$0000
dc.w    $01b2,$0000
dc.w    $01b4,$0000
dc.w    $01b6,$0000
dc.w    $01b8,$0000
dc.w    $01ba,$0000
dc.w    $01bc,$0000
dc.w    $01be,$0000
```

```
dc.w	$0106,%1110001001000000	;Bank 7
dc.w	$0180,$0000		;2nd 32 colors, LSB
dc.w	$0182,$0000
dc.w	$0184,$0000
dc.w	$0186,$0000
dc.w	$0188,$0000
dc.w	$018a,$0000
dc.w	$018c,$0000
dc.w	$018e,$0000
dc.w	$0190,$0000
dc.w	$0192,$0000
dc.w	$0194,$0000
dc.w	$0196,$0000
dc.w	$0198,$0000
dc.w	$019a,$0000
dc.w	$019c,$0000
dc.w	$019e,$0000
dc.w	$01a0,$0000
dc.w	$01a2,$0000
dc.w	$01a4,$0000
dc.w	$01a6,$0000
dc.w	$01a8,$0000
dc.w	$01aa,$0000
dc.w	$01ac,$0000
dc.w	$01ae,$0000
dc.w	$01b0,$0000
dc.w	$01b2,$0000
dc.w	$01b4,$0000
dc.w	$01b6,$0000
dc.w	$01b8,$0000
dc.w	$01ba,$0000
dc.w	$01bc,$0000
dc.w	$01be,$0000
```

BEENDEN DER COPPERLISTE

Den Abschluß unserer Copperliste bildet der Wait-Befehl auf die letzte NTSC Linie.

```
        dc.w    $ff07,$fffe     ;Wait for last NTSC line
NoCop:  dc.w    $ffff,$fffe     ;End of copperlist
```

Mit der oben beschriebenen Copperliste sollte es nun keine Probleme geben, wenn Sie Ihre Programme auch für die neuen Amigas adaptieren wollen.

5. PRAKTISCHE PROGRAMMBEISPIELE

5.1. LAUFSCHRIFTEN

Im folgenden Kapitel wollen wir anhand einiger praktischer Beispiele die Funktionsweise der einzelnen Bausteine (z.B.: Blitter, Copper,...) näher beleuchten. Sehr gut kann man den Einsatz anhand von Laufschriften demonstrieren. Einige ausgewählte Beispiele zeigen, daß selbst schwierige Demos oft gar nicht so schwer zu realisieren sind.

5.1.1. SINUS-LAUFSCHRIFT

Als nächstes wollen wir uns einer speziellen Laufschrift-Art widmen, die in diversen Intros und Demos häufig verwendet wird, und der man nachsagt, daß sie ungemein schwer zu programmieren sei. Daß dem nicht so ist, wollen wir auf den folgenden Seiten beweisen.

Die Schwierigkeit bei dieser Laufschrift liegt eigentlich nicht in der Art der Programmierung des Blitters, sondern eher in der Berechnung der Darstellung. Wie das zu verstehen ist, werden wir noch näher erläutern. Sehen wir uns doch vorher die Aufgabenstellung an. Es soll eine Laufschrift von rechts nach links über den Bildschirm gescrollt werden, wobei sich die Buchstaben einer Sinuskurve anschmiegen (siehe Abbildung).



Im Prinzip funktioniert das so: Zuerst lassen wir eine ganz normale Laufschrift durch den Speicher wandern, nur mit dem Unterschied, daß der Betrachter diese nicht sieht.

Diese nicht sichtbare durch den Speicher wandernde Laufschrift wird durch ein kurzes Unterprogramm auf der sichtbaren Bitplane an eine Sinuskurve angeschmiegt.

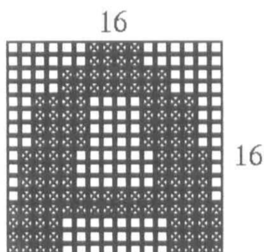
Nun die einzelnen Schritte, die für das Programm nötig sind, in einer kurzen Übersicht:

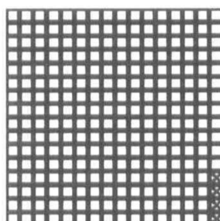
1) Bildschirm löschen.
2) Sinus darstellen.
3)Einfache Laufschrift scrollen und neuen Buchstaben darstellen.

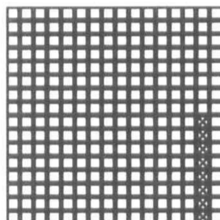
Der schwierigste Teil hierbei ist die Routine, die die einfache Laufschrift an die Sinuskurve angleicht.

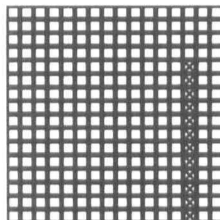
Dazu verwenden wir zwei Quellen, A und B, wobei in A die Adresse der einfachen Laufschrift eingetragen wird, und in B die Adresse der Zielbitplane verwendet wird, die auch im Zielregister D steht. Die Zielbitplane muß deshalb als Quelle eingetragen werden, damit der Hintergrund nicht zerstört wird. Denn jeder Buchstabe wird spaltenweise (siehe unten stehende Abbildung) auf die Plane geblittet. Dadurch würde der Hintergrund links und rechts von der Spalte im Bereich des zu blittenden Words zerstört werden. Um dies zu verhindern, wird der Hintergrund ebenfalls als Quelle initialisiert und mitkopiert.

Die Abbildung auf der nächsten Seite zeigt, wie ein Buchstabe mit Hilfe der Maske in Spalten zerlegt wird und somit verschoben werden kann.



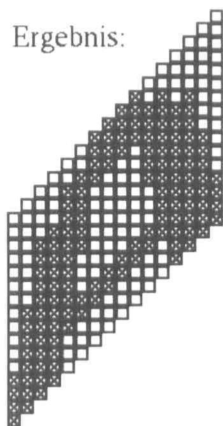






u.s.w.

Ergebnis:



Das unten stehende Programmstück erledigt die Aufteilung der einzelnen Buchstaben in Spalten und weist jeder Spalte die exakte Position auf der Bitplane zu:

```
;
; Sinus blitten
;
        move.w  #48,BLTAMOD(a6)
        move.w  #38,BLTBMOD(a6)
        move.w  #38,BLTDMOD(a6)
```

```
        move.w  #%0000110111111100,BLTCON0(a6)
        clr.w   BLTCON1(a6)
        move.w  #$FFFF,BLTALWM(a6)
;
        moveq   #38,d4                  ;wort-positionszähler
        lea     sinus(pc),a2
        adda.l  Sinc,a2
;
        move.l  #$DFF044,a0
        move.l  #$DFF058,a1
        move.l  #$DFF050,a3
        move.l  #$DFF04c,a4
        move.l  #$DFF054,a6
;
        move.l  #pic,d2
        move.l  #Space+40,d0
        move.w  #1025,d5 ;BLTSIZE
        move.w  #1,d6
;
        add.l   #4,Sinc
        cmp.l   #504,Sinc
        blt.s   NoEnd2
        clr.l   Sinc
NoEnd2:
;
        moveq   #%1,d3                  ; Maske
NoEnd3:
;
        move.l  d2,d1
        add.w   d4,d1
        add.w   (a2)+,d1
;
BS:     btst    #6,$dff002
        bne.s   BS
;
        move.w  d3,(a0)
```

```
        move.l  d0,(a3)
;
        move.l  d1,(a4)
        move.l  d1,(a6)
;
        move.w  d5,(a1)
;
        rol.w   #1,d3           ;Pixelanz.
        cmp.w   d6,d3           ;16/Pixelanz.
        bne.s   NoEnd3
;
        subq.w  #2,d0
        subq.w  #2,d4           ;Linker Rand
        bge.s   NoEnd3
;
    rts
```

Zuerst werden die notwendigen Modulo-Werte initialisiert, sowie Quelle A, B und das Ziel D angewählt. Weiterhin wird die Miniterm-Kombination eingetragen. In diesem Fall lautet die Kombination %11111100, da das Ergebnis 1 sein soll, wenn entweder A oder B gleich 1 ist (A = 1 oder B = 1 -> Ergebnis = 1).

Die Sinus-Daten, sowie diejenigen Register, welche innerhalb der Schleife verwendet werden, werden in Adressregister geschrieben, da der Zugriff auf ein Adressregister schneller ist, als direkt auf die Hardwareregister. Dies hilft mit, Zeit zu sparen.
Im Label Space ist die einfache Laufschrift enthalten, die unsichtbar durch den Speicher wandert. Von dort holen wir das aktuelle Zeichen und blitten es spaltenweise in den sichtbaren Bereich. Um jedes einzelne Zeichen in die wie oben erläuterten Spalten zu zerlegen, verwenden wir das Register BLTAFWM, welches am Anfang der Routine auf 1 (entspricht der ersten Spalte) gesetzt wird. Im Laufe der Schleife wird dieser Einser um ein Bit nach links verschoben. Diese Schleife wird insgesamt 16 Mal durchlaufen, bis alle Spalten fertig geblittet wurden.

Die Y-Verschiebung der Spalten wird durch die Werte, die unter dem Label Sinus gespeichert sind, erreicht.

Hier wurde ebenfalls mit einem kleinen Trick gearbeitet: Normalerweise enthält diese Sinustabelle einfache Sinus-Werte, wie man sie mittels eines kurzen Basicprogramms berechnen kann. Diesen Sinus-Werten müssen während des Programmablaufes die richtigen Bildschirmpositionen zugeordnet werden - das kostet aber viel wertvolle Rechenzeit. Daher ordnen wir die Bildschirmpositionen dieser Tabelle schon am Anfang des Programms mittels der folgenden Routine zu.

```
SinusChange:
;
          lea       sinus(pc),a2
          lea       sinus(pc),a3
;
          move.l    #1007,d3
NoEnd4:
          move.w    (a2)+,d0
          mulu      #40,d0
          move.w    #5280,d1
          sub.w     d0,d1
          move.w    d1,(a3)+
;
          dbf       d3,NoEnd4
;
          rts
```

Die auf der Beispiel-Diskette enthaltenen Sinus-Werte werden in unserem Listing viermal hintereinander benötigt, daher müssen sie auch viermal eingelesen werden.

```
;           Sinus-Laufschrift
;
;           Nachzuladende Files:
;           "df0:raw/16x16_1pl.raw"         auf "font"
;           "df0:SinusDat.tab"              auf "sinus"
;           "df0:SinusDat.tab"              auf "sinus"+504
;           "df0:SinusDat.tab"              auf "sinus"+1008
;           "df0:SinusDat.tab"              auf "sinus"+1512
;
dmaconr  =          $02
bltddat  =          $00
bltcon0  =          $40
bltcon1  =          $42
bltafwm  =          $44
bltalwm  =          $46
bltcpth  =          $48
bltapth  =          $50
bltaptl  =          $52
bltdpth  =          $54
bltsize  =          $58
bltcmod  =          $60
bltbmod  =          $62
bltamod  =          $64
bltdmod  =          $66
bltcdat  =          $70
bltbdat  =          $72
bltadat  =          $74
;
s:          bsr.L     opening
            bsr.L     initcolors
;
            bsr.L     SinusChange
;
            lea       Text(pc),a5
            move.l    a5,Store
            move.b    #0,Counter
```

```
            move.l      #0,Sinc
;
;
loop:       move.l      $dff004,d0
            cmp.l       #$0d000,d0
            bne.s       loop
;
;
            move.l      #$dff000,a6
            bsr.L       BlitSinus
            move.l      #$dff000,a6
            bsr.L       Scroll
;
;
            btst        #6,$bfe001
            bne.s       loop
;
;
end:        move.w      #$c000,$dff09a
            moveq       #0,d0
e:          rts
;
opening:    move.w      #$4000,$dff09a
            move.l      #copperl,$dff084
            move.l      #pic,d0
            move        d0,pl1+6
            swap        d0
            move        d0,pl1+2
            swap        d0
;
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.      l #7,d1
initsprites: swap       d0
            move        d0,(a0)
            add.l       #4,a0
```

```
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
initcolors: lea         color(pc),a0
            lea         colortable(pc),a1
            moveq       #1,d0
            move        #$180,d1
copycolors:
            move        d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
;
            rts
;
colortable: dc.w        $0000,$0fff
;
copperl:    dc.w        $008e,$3090
            dc.w        $0090,$f8f0
            dc.w        $0092,$0038
            dc.w        $0094,$00d0
            dc.w        $0102,$0000
            dc.w        $0104,$0024
            dc.w        $0108,$0000
            dc.w        $010a,$0000
            dc.w        $0100,$1200
pl1:        dc.w        $00e0,$0000
            dc.w        $00e2,$0000
pl2:        dc.w        $00e4,$0000
            dc.w        $00e6,$0000
pl3:        dc.w        $00e8,$0000
            dc.w        $00ea,$0000
pl4:        dc.w        $00ec,$0000
```

```
        dc.w      $00ee,$0000
pl5:    dc.w      $00f0,$0000
        dc.w      $00f2,$0000
pl6:    dc.w      $00f4,$0000
        dc.w      $00f6,$0000
spl:    dc.w      $0120,$0000
        dc.w      $0122,$0000
        dc.w      $0124,$0000
        dc.w      $0126,$0000
        dc.w      $0128,$0000
        dc.w      $012a,$0000
        dc.w      $012c,$0000
        dc.w      $012e,$0000
        dc.w      $0130,$0000
        dc.w      $0132,$0000
        dc.w      $0134,$0000
        dc.w      $0136,$0000
        dc.w      $0138,$0000
        dc.w      $013a,$0000
        dc.w      $013c,$0000
        dc.w      $013e,$0000
color:  blk.l     4,0
        dc.w      $ffff,$fffe
;
sprite: dc.l      0
size:   dc.l      8000
;
;******************************
;
Scroll: btst      #6,$dff002
        bne.s     Scroll
;
        move.w    #%1111100111110000,BLTCON0(a6)
        clr.w     BLTCON1(a6)
;
```

```
        move.w  #4,BLTAMOD(a6)
        move.w  #4,BLTDMOD(a6)
        move.w  #$FFFF,BLTAFWM(a6)
        move.w  #$FFFF,BLTALWM(a6)
;
        move.l  #Space,BLTDPTH(a6)
        move.l  #Space+2,BLTAPTH(a6)
        move.w  #1047,BLTSIZE(a6)
;
; Neuer Buchstabe wird auf Plane kopiert
;
        add.b   #1,Counter
        cmp.b   #16,Counter
        bne.s   NoLetter
        clr.b   Counter
;
        move.l  Store,a5
        move.b  (a5)+,d1
        move.l  a5,Store
;
        cmp.b   #"o",d1
        bne.s   NoEnd
        move.l  #Text,store
        rts
;
NoEnd:  sub.b   #32,d1
        ext     d1
        bra.s   CopyLetter
;
NoLetter:       rts
;
;************************************
;
CopyLetter:
        btst    #6,$dff002
        bne.s   CopyLetter
```

```
;
            move.w    #%0000100111110000,BLTCON0(a6)
            clr.w     BLTCON1(a6)
;
            move.w    #38,BLTAMOD(a6)
            move.w    #48,BLTDMOD(a6)
;
            move.l    #Font,d0
;
            asl.w     #2,d1
            lea       Letter(pc),a3
            move.l    (a3,d1.w),d2
            add.l     d2,d0
            move.l    d0,BLTAPTH(a6)
;
            move.l    #Space+42,BLTDPTH(a6)
;
            move.w    #1025,BLTSIZE(a6)
;
            rts
;
BlitSinus:  btst      #6,$dff002
            bne.s     BlitSinus
; Löschen
            clr.w     BLTDDAT(a6)
            clr.w     BLTADAT(a6)
;
            move.w    #%0000000111110000,BLTCON0(a6)
            clr.w     BLTCON1(a6)
;
            clr.w     BLTAMOD(a6)
            clr.w     BLTDMOD(a6)
;
            move.w    #$FFFF,BLTAFWM(a6)
            move.w    #$FFFF,BLTALWM(a6)
;
```

```
          move.l    #pic,BLTAPTH(a6)
          move.l    #pic+1680,BLTDPTH(a6)
;
          move.w    #6804,BLTSIZE(a6)
;
WaitLoop: btst      #6,DMACONR(a6)
          bne.s     WaitLoop
;
; Sinus blitten
;
          move.w    #48,BLTAMOD(a6)
          move.w    #38,BLTBMOD(a6)
          move.w    #38,BLTDMOD(a6)
;
          move.w    #%0000110111111100,BLTCON0(a6)
          clr.w     BLTCON1(a6)
          move.w    #$FFFF,BLTALWM(a6)
;
          moveq     #38,d4                  ;wort-positionszähler
          lea       sinus(pc),a2
          adda.l    Sinc,a2
;
          move.l    #$DFF044,a0
          move.l    #$DFF058,a1
          move.l    #$DFF050,a3
          move.l    #$DFF04c,a4
          move.l    #$DFF054,a6
;
          move.l    #pic,d2
          move.l    #Space+40,d0
          move.w    #1025,d5 ;BLTSIZE
          move.w    #1,d6
;
          add.l     #4,Sinc
          cmp.l     #504,Sinc
          blt.s     NoEnd2
```

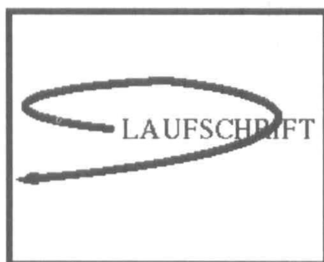
```
        clr.l     Sinc
NoEnd2:
;
        moveq     #%1,d3              ; Maske
NoEnd3:
;
        move.l    d2,d1
        add.w     d4,d1
        add.w     (a2)+,d1
;
BS:     btst      #6,$dff002
        bne.s     BS
;
        move.w    d3,(a0)
;
        move.l    d0,(a3)
;
        move.l    d1,(a4)
        move.l    d1,(a6)
;
        move.w    d5,(a1)
;
        rol.w     #1,d3               ;Pixelanz.
        cmp.w     d6,d3               ;16/Pixelanz.
        bne.s     NoEnd3
;
        subq.w    #2,d0
        subq.w    #2,d4               ;Linker Rand
        bge.s     NoEnd3

        rts
;
;*****************************
;
```

```
SinusChange:
            lea         sinus(pc),a2
            lea         sinus(pc),a3
;
            move.l      #1007,d3
NoEnd4:
            move.w      (a2)+,d0
            mulu        #40,d0
            move.w      #5280,d1
            sub.w       d0,d1
            move.w      d1,(a3)+
            dbf         d3,NoEnd4
;
            rts
;
;*****************************
;
Counter:                dc.l 0
Sinc:                   dc.l 0
Store:                  dc.l 0
Text:                   dc.b "SINUS - LAUFSCHRIFT !!!"
dc.b        "      o"
even
;
Font:                   blk.b 2000,0
Space:                  blk.b 1200,0
Sinus:                  blk.b 2016,0
;
Letter:
dc.l 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38
dc.l 640,642,644,646,648,650,652,654,656,658,660,662,664,666
dc.l 668,670,672,674,676,678
dc.l 1280,1282,1284,1286,1288,1290,1292,1294,1296,1298
dc.l 1300,1302,1304,1306,1308,1310,1312,1314,1316,1318
;
pic:        blk.b 8000,0
```

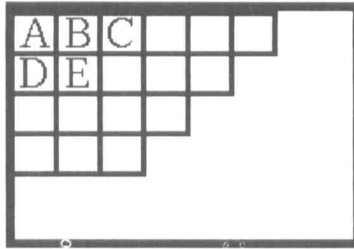
5.1.2. LAUFSCHRIFT NACH KOORDINATENTABELLE

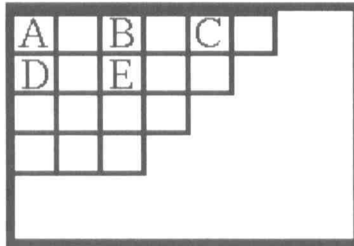
Die nächste Laufschrift wirkt ebenfalls sehr überraschend auf den Betrachter. Er glaubt zuerst, eine einfache Laufschrift, die sich von rechts nach links bewegt, vor sich zu haben. Aber dem ist nicht so, da die Buchstaben, wenn sie fast beim linken Rand angekommen sind, eine große Kurve nach rechts drehen, um nach einer großen Kreisbewegung wieder weiter links unten zu verschwinden.



Einen kleinen Nachteil hat diese Art der Laufschriftgestaltung schon: Dadurch, daß jeder einzelne Buchstabe als eigenes Bob behandelt werden muß, ist es leider bei so vielen Buchstaben nicht möglich, einen 32 farbigen Font darzustellen. Aber um etwas Farbe ins Spiel zu bringen, könnte man in Hintergrund eine Farb-Copperliste erstellen (siehe dazu Anhang 7.5. Anregungen).

Das Prinzip dieser Laufschrift ist ziemlich einfach: Man scrollt nicht den gesamten Bereich mit allen darauf befindlichen Buchstaben, wie wir es bis jetzt bei den anderen Laufschriften getan haben, sondern, wie schon erwähnt, wird jeder Buchstabe als eigenes Bob (Blitter-Object) behandelt. Dies hat aber ebenfalls einen kleinen Nachteil: Normalerweise kann man mit dem Blitter, wie wir bereits wissen, nur an Wordadressen kopieren. Also ist ein Kopieren an jede beliebige Adresse nicht möglich. Daher bedient man sich des Blitter-Shift Registers. Das bedeutet, daß man die Bits links hinausschiebt, dies hat aber den Nachteil, daß jene Bits, die rechts des Buchstabens liegen, ebenfalls nach links geschoben werden.





Normalerweise liegen die Buchstaben immer 16 Punkte weit entfernt nebeneinander. Wenn man nun nach jedem Buchstaben weitere 16 Pixel Platz läßt, werden, durch das Rechtsschieben der einzelnen Buchstaben, keine ungewünschten Pixel mitgeschoben, sondern nur der neben jedem Buchstaben liegende frei Platz.

Um die lästige und zeitaufwendige Berechnung der Bildschirmposition der Buchstaben während des Programmablaufes zu berechnen, wird dies mit einer kurzen Routine gleich nach dem Programmstart erledigt:

```
ChangeCoords:
        lea     oords(pc),a0
        lea     oords2(pc),a1
;
NoEnd2:
        move.l  #Pic,d0
;
; Y-Koordinate berechnen
```

```
;
        clr.l       1
        move.w      $2(a0),d1
        mulu        44,d1
        add.l       1,d0
;
; X-Koordinate berechnen
;
        move.w      (a0),d1
        and.w       $f,d1
        ror.w       4,d1
        move.w      #%0000110111111100,d2 ; für BLTCON0
        add.w       1,d2
        move.w      d2,(a1)+
;
        clr.l       1
        move.w      (a0),d1
        and.w       $fff0,d1
        asr.w       3,d1
        add.l       1,d0
;
        move.l      d0,(a1)+ ;; für BLTDPTH und BLTBPTH
;
        add.l       4,a0
        cmp.l       Coords+1464,a0
        bne.s       NoEnd2
;
        rts
```

Die Bewegungen, welche die Buchstaben durchführen sollen, werden am besten mit einem kurzen Basicprogramm berechnet. Dieses Basicprogramm liefert uns aber nur das Ergebnis einer mathematischen Kurve (Sie findes dieses als File auf der mitgelieferten Diskette unter dem Namen "Kords.tab"), aber noch nicht die endgültige Bildschirmposition, an der ein Buchstabe plaziert werden soll.

Daher ordnet die obige Routine jedem X- und Y-Zahlenwert aus dem nachgeladenen File die richtige Bildschirmposition zu und speichert diese unter dem Label "Coords2" ab. Das Hauptprogramm muß also lediglich die Bildschirmposition aus dieser neu erstellten Tabelle nehmen und den aktuellen Buchstaben an diese Adresse "blitten".

```
;
;           Koordinaten-Laufschrift
;
;Nachzuladende Files:
;             "df0:raw/16x16_1pl-breit.raw"   auf "font"
;             "df0:Kords.tab"                 auf "Coords"
;
dmaconr   =        $02
bltddat   =        $00
bltcon0   =        $40
bltcon1   =        $42
bltafwm   =        $44
bltalwm   =        $46
bltcpth   =        $48
bltbpth   =        $4c
bltapth   =        $50
bltaptl   =        $52
bltdpth   =        $54
bltsize   =        $58
bltcmod   =        $60
bltbmod   =        $62
bltamod   =        $64
bltdmod   =        $66
bltcdat   =        $70
bltbdat   =        $72
bltadat   =        $74
;
;
s:        bsr.s    opening
```

```
            bsr.L       initcolors
            bsr.L       ChangeCoords
;
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$0b000,d0
            bne.s       loop
;
            move.l      #$dff000,a6
            bsr.L       copyletter
;
            btst        #6,$bfe001
            bne.s       loop
;
;
end:        move.w      #$c000,$dff09a
            moveq       #0,d0
e:          rts
;
opening:    move.w      #$4000,$dff09a
            move.l      #copperl,$dff084
            move.l      #Pic,d0
            add.l       #2,d0
            move        d0,pl1+6
            swap        d0
            move        d0,pl1+2
;
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.l     #7,d1
initsprites: swap       d0
            move        d0,(a0)
            add.l       #4,a0
            swap        d0
            move        d0,(a0)
```

```
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
initcolors: lea         color(pc),a0
            lea         colortable(pc),a1
            moveq       #1,d0
            move        #$180,d1
copycolors:
            move        d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
;
            rts
;
colortable:
            dc.w        $0000,$0fff
;
copperl:
            dc.w        $008e,$3090
            dc.w        $0090,$f8f0
            dc.w        $0092,$0038
            dc.w        $0094,$00d0
            dc.w        $0102,$0000
            dc.w        $0104,$0024
            dc.w        $0108,$0004
            dc.w        $010a,$0004
            dc.w        $0100,$1200
pl1:        dc.w        $00e0,$0000
            dc.w        $00e2,$0000
sp1:        dc.w        $0120,$0000
            dc.w        $0122,$0000
            dc.w        $0124,$0000
            dc.w        $0126,$0000
            dc.w        $0128,$0000
```

```
            dc.w        $012a,$0000
            dc.w        $012c,$0000
            dc.w        $012e,$0000
            dc.w        $0130,$0000
            dc.w        $0132,$0000
            dc.w        $0134,$0000
            dc.w        $0136,$0000
            dc.w        $0138,$0000
            dc.w        $013a,$0000
            dc.w        $013c,$0000
            dc.w        $013e,$0000
color:      blk.l       2,0
            dc.w        $ffff,$fffe
;
;*************************************
;
ChangeCoords:
            lea         Coords(pc),a0
            lea         Coords2(pc),a1
;
NoEnd2:
            move.l      #Pic,d0
;
; Y-Koordinate berechnen
;
            clr.l       d1
            move.w      $2(a0),d1
            mulu        44,d1
            add.l       d1,d0
;
; X-Koordinate berechnen
;
            move.w      (a0),d1
            and.w       #$f,d1
            ror.w       #4,d1
            move.w      #%0000110111111100,d2  ; für BLTCON0
```

```
            add.w       d1,d2
            move.w      d2,(a1)+
;
            clr.l       d1
            move.w      (a0),d1
            and.w       #$fff0,d1
            asr.w       #3,d1
            add.l       d1,d0
;
            move.l      d0,(a1)+   ; für BLTDPTH und BLTBPTH
;
            add.l       #4,a0
            cmp.l       #Coords+1464,a0
            bne.s       NoEnd2
;
            rts
;
;*********************************
;
CopyLetter:
;
; Löschen
;
            clr.w       BLTADAT(a6)
            clr.w       BLTDDAT(a6)
            move.w      #%0000000100000000,BLTCON0(a6)
            clr.w       LTCON1(a6)
            move.w      #40,BLTDMOD(a6)
            move.l      #2160,d5
;
            lea         Coords2(pc),a0
            add.l       CoordsCount,a0
NoEnd3:
            btst        #6,DMACONR(a6)
            bne.s       NoEnd3
;
```

```
        move.l   $2(a0,d5.l),BLTDPTH(a6)
        move.w   #1026,BLTSIZE(a6)
;
        sub.l    #30,d5               ;Abstand der Buch-
                                      ;staben in Tabellen-
        bne.s    NoEnd3               ;einträgen mal sechs
;
; Buchstaben blitten
;
        add.l    #6,CoordsCount       ; Alle wieviel Einträge
                                      ; mal sechs der
        cmp.l    #30,CoordsCount      ; Koordinatenzähler und
                                      ; der Textzähler
        bne.s    NoZero               ; herabgesetzt werden
        clr.l    CoordsCount
        add.l    #1,TextCount
        cmp.b    #$FF,(a5)
        bne.s    NoZero
        clr.l    TextCount
NoZero:
        btst     #6,DMACONR(a6)
        bne.s    NoZero
;
        clr.w    BLTCON1(a6)
        move.w   #36,BLTAMOD(a6)
        move.w   #40,BLTDMOD(a6)
        move.w   #40,BLTBMOD(a6)
        move.w   #$ffff,BLTAFWM(a6)
        move.w   #$ffff,BLTALWM(a6)
;
        lea      Letter(pc),a3
        lea      Text(pc),a5
        lea      Coords2(pc),a0
        add.l    TextCount,a5
        add.l    CoordsCount,a0
;
```

```
        move.l  #2160,d5
NoEnd:
        move.l  #Font,d0
        clr.l   d1
        move.b  (a5)+,d1
        sub.b   #32,d1
        ext     d1
        asl.w   #2,d1
        move.l  (a3,d1.w),d2
        add.l   d2,d0
;
Wait2:  btst    #6,DMACONR(a6)
        bne.s   Wait2
;
        move.l  d0,BLTAPTH(a6)
        move.w  $0(a0,d5.l),BLTCON0(a6)
        move.l  $2(a0,d5.l),BLTDPTH(a6)
        move.l  $2(a0,d5.l),BLTBPTH(a6)
        move.w  #1026,BLTSIZE(a6)
;
        sub.l   #30,d5          ;Abstand der Buch-
                                ;staben in Tabellen-
        bne.s   NoEnd           ;einträgen mal sechs
;
       rts
;
;****************************************
;
even
text: dc.b "                                        "
 dc.b "                    "
 dc.b
"DIESER TEXT WIRD NACH EINER FREI WAEHLBAREN "
 dc.b "KOORDINATENTABELLE BEWEGT!!!"
 dc.b "                                        "
 dc.b "                    ",$FF
```

```
 even
;
sprite:                 dc.l 0
CoordsCount:            dc.l 0
TextCount:              dc.l 0
;
Letter:
  dc.l 0,4,8,12,16,20,24,28,32,36
  dc.l 640,644,648,652,656,660,664,668,672,676
  dc.l 1280,1284,1288,1292,1296,1300,1304,1308,1312,1316
  dc.l 1920,1924,1928,1932,1936,1940,1944,1948,1952,1956
  dc.l 2560,2564,2568,2572,2576,2580,2584,2588,2592,2596
  dc.l 3200,3204,3208,3212,3216,3220,3224,3228,3232,3236
;
Font:                   blk.b 3840,0
Coords:                 blk.b 1464,0
Coords2:                blk.b 2196,0
Pic:          blk.b 8800,0
;
```

5.1.3. JUMP-LAUFSCHRIFT

Als Abschluß des Laufschriftkapitels wollen wir uns mit einem vor allem für die Videonachbearbeitung gut geeigneten Beispiel einer Laufschrift befassen.

Die nun folgende Laufschrift besitzt eine Größe von 32 mal 32 Punkten und soll, während sie von rechts nach links über den Bildschirm wandert, auf und ab springen.



Damit man auch die Farbenvielfalt ausnützen kann, besitzt der Font 5 Bitplanes, also 32 verschiedene Farben. Wiederum muß der mit einem Malprogramm gestaltete Schriftsatz, falls sie selber einen gestalten wollen, zuerst ins RAW-Format umgewandelt werden und anschließend müssen die einzelnen Bitplanes mit dem Bitplane-Mixer nebeneinander gelegt werden, damit man sie in diesem Beispiel verwenden kann. Oder Sie verwenden den fertig konvertierten Font, der auf der Beispieldiskette enthalten ist.

Um eine Laufschrift auf und ab springen zu lassen, gibt es mehrere Möglichhkeiten:

1) Man könnte den gesamten Bildschirm scrollen, indem man die Planeadressen ständig ändert.
2) Eine wesentlich elegantere Methode ist, daß man einen Wait-Befehl vor das Register $dff100 setzt, das zuständig ist für das Einschalten der Playfields. Anschließend ändert man diesen Wait-Befehl und schon springt die Laufschrift.

Wie das nun programmtechnisch aussieht, sehen wir uns anhand des Listings an:

```
copperl:
        dc.w    $008e,$3064
        dc.w    $0090,$f8d1
```

```
        dc.w    $0092,$0020
        dc.w    $0094,$00d8
        dc.w    $0102,$0000
        dc.w    $0104,$0000
        dc.w    $0108,$0004
        dc.w    $010a,$0004
```

Zuerst werden die für die Bilddarstellung wichtigen Register initialisiert.

```
        dc.w    $0100,$0200
```

In dieser Zeile werden die Bildplanes ausgeschalten.

```
pl1:    dc.w    $00e0,$0000
        dc.w    $00e2,$0000
pl2:    dc.w    $00e4,$0000
        dc.w    $00e6,$0000
pl3:    dc.w    $00e8,$0000
        dc.w    $00ea,$0000
pl4:    dc.w    $00ec,$0000
        dc.w    $00ee,$0000
pl5:    dc.w    $00f0,$0000
        dc.w    $00f2,$0000
sp1:    dc.w    $0120,$0000
        dc.w    $0122,$0000
        dc.w    $0124,$0000
        dc.w    $0126,$0000
        dc.w    $0128,$0000
        dc.w    $012a,$0000
        dc.w    $012c,$0000
        dc.w    $012e,$0000
        dc.w    $0130,$0000
        dc.w    $0132,$0000
        dc.w    $0134,$0000
        dc.w    $0136,$0000
```

```
         dc.w    $0138,$0000
         dc.w    $013a,$0000
         dc.w    $013c,$0000
         dc.w    $013e,$0000
color:   blk.l   32,0
```

Hier werden die Bitplanes, sowie die Sprites und die Farbregister eingetragen, die dann von der Routine "opening" initialisiert werden.

```
cins1:   dc.w    $50df,$fffe
         dc.w    $0100,$5200
```

Nun folgt der Wait-Befehl, von dem oben gesprochen wurde. Dieser Befehl wird später vom Programm aus ständig verändert. Nach dem Wait folgt dann das Kommando zum Einschalten der oben initialisierten Bitplanes.

```
cins2:   dc.w    $700f,$fffe
         dc.w    $0100,$0200
         dc.w    $ffff,$fffe
```

Der zweite Wait schaltet die Bitplanes kanpp unter der Laufschrift wieder ab.

Verändert werden die beiden Wait-Befehle mit der folgenden Routine:

```
CopperChange:
         lea     Sinus(pc),a0
         add.l   SinusCount,a0
;
         move.w  #$68,d1
         add.b   $1(a0),d1
         move.b  d1,Cins1
         add.b   #$20,d1
         move.b  d1,Cins2
```

```
            add.l     #8,SinusCount
            cmp.l     #504,SinusCount
            bne.s     NoEnd2
            clr.l     SinusCount
NoEnd2:     rts
```

Damit die Bewegung der Laufschrift nicht linear ist, sondern ein "Springen" der Schrift dargestellt wird, verwenden wir dazu Sinus-Daten, die zuerst von der mitgelieferten Beispieldiskette nachgeladen werden müssen.

Wie man die Buchstaben einer Laufschrift auf dem Bildschirm bewegt, wollen wir an dieser Stelle nicht mehr erklären, da dies in den vorhergehenden Kapiteln schon ausführlich besprochen wurde. Falls nötig, schlagen Sie bitte weiter vorne nach.

Hier nun das fertige Listing:

```
;           Jump-Laufschrift
;
;             Nachzuladende Files:
;             "SinusDat.tab"              auf "sinus"
;             "raw/32x32_5pl.con"         auf "font"
;
dmaconr   =     $02
bltddat   =     $00
bltcon0   =     $40
bltcon1   =     $42
bltafwm   =     $44
bltalwm   =     $46
bltcpth   =     $48
bltbpth   =     $4c
bltapth   =     $50
bltaptl   =     $52
bltdpth   =     $54
bltsize   =     $58
```

```
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
s:          bsr.s       opening
            bsr.L       initcolors
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$0f000,d0
            bne.s       loop
;
            move.l      #$DFF000,a6
            bsr.L       Scroll
            bsr.L       CopperChange
;
            btst        #6,$bfe001
            bne.s       loop
;
end:        move.w      #$c000,$dff09a
e:          moveq.l     #0,d0
            rts
;
opening:    move.w      #$4000,$dff09a
            move.l      #copperl,$dff084
            move.l      #Pic,d0
            move        d0,pl1+6
            swap        d0
            move        d0,pl1+2
            swap        d0
            add.l       size,d0
            move        d0,pl2+6
```

```
            swap      d0
            move      d0,pl2+2
            swap      d0
            add.l     size,d0
            move      d0,pl3+6
            swap      d0
            move      d0,pl3+2
            swap      d0
            add.l     size,d0
            move      d0,pl4+6
            swap      d0
            move      d0,pl4+2
            swap      d0
            add.l     size,d0
            move      d0,pl5+6
            swap      d0
            move      d0,pl5+2
;
            move.l    #sprite,d0
            lea       sp1+2(pc),a0
            moveq.l   #7,d1
initsprites: swap     d0
            move      d0,(a0)
            add.l     #4,a0
            swap      d0
            move      d0,(a0)
            add.l     #4,a0
            dbf       d1,initsprites
            rts
;
initcolors: lea       color(pc),a0
            lea       font+38400,a1
            moveq     #31,d0
            move      #$180,d1
copycolors:
            move      d1,(a0)+
```

```
        move    (a1)+,(a0)+
        addq    #2,d1
        dbf     d0,copycolors
;
        rts
;
copperl: dc.w   $008e,$3064
        dc.w    $0090,$f8d1
        dc.w    $0092,$0020
        dc.w    $0094,$00d8
        dc.w    $0102,$0000
        dc.w    $0104,$0000
        dc.w    $0108,$0004
        dc.w    $010a,$0004
        dc.w    $0100,$0200
pl1:    dc.w    $00e0,$0000
        dc.w    $00e2,$0000
pl2:    dc.w    $00e4,$0000
        dc.w    $00e6,$0000
pl3:    dc.w    $00e8,$0000
        dc.w    $00ea,$0000
pl4:    dc.w    $00ec,$0000
        dc.w    $00ee,$0000
pl5:    dc.w    $00f0,$0000
        dc.w    $00f2,$0000
sp1:    dc.w    $0120,$0000
        dc.w    $0122,$0000
        dc.w    $0124,$0000
        dc.w    $0126,$0000
        dc.w    $0128,$0000
        dc.w    $012a,$0000
        dc.w    $012c,$0000
        dc.w    $012e,$0000
        dc.w    $0130,$0000
        dc.w    $0132,$0000
        dc.w    $0134,$0000
```

```
            dc.w      $0136,$0000
            dc.w      $0138,$0000
            dc.w      $013a,$0000
            dc.w      $013c,$0000
            dc.w      $013e,$0000
color:      blk.l     32,0
;
cins1:      dc.w      $50df,$fffe
            dc.w      $0100,$5200
;
cins2:      dc.w      $700f,$fffe
            dc.w      $0100,$0200
            dc.w      $ffff,$fffe
;
;*****************************************
Scroll:
;
; Gesamten Scrollbereich um 1 Pixel nach links shiften
;
            move.w    #%1110100111110000,BLTCON0(A6)
            clr.w     BLTCON1(a6)
            clr.w     BLTAMOD(a6)
            clr.w     BLTDMOD(a6)
            move.w    #$FFFF,BLTAFWM(a6)
            move.w    #$FFFF,BLTALWM(a6)
            move.l    #Pic+2,BLTAPTH(a6)
            move.l    #Pic,BLTDPTH(a6)
            move.w    #10266,BLTSIZE(a6)
;
WaitLoop:   btst      #6,DMACONR(A6)
            bne.s     WaitLoop
;
;Jedes 16.Mal ein neuen Buchstaben an den rechten Rand blitten
;
            add.w     #1,LetterMark
            cmp.w     #16,LetterMark
```

```
        bne.L     NoLetter
        clr.w     LetterMark
;
        move.w    #%0000100111110000,BLTCON0(A6)
        clr.w     BLTCON1(a6)
        move.w    #196,BLTAMOD(a6)
        move.w    #48,BLTDMOD(a6)
        move.w    #$FFFF,BLTAFWM(a6)
        move.w    #$FFFF,BLTALWM(a6)
;
        clr.l     d6
        clr.l     d7
Nochmal: move.l   #Font,d0
        add.l     d6,d0
        move.l    #Text,a0
        add.l     TextMark,a0
        clr.l     d1
        move.b    (a0),d1
        cmp.b     #$FF,d1
        bne.s     NoEnd
        clr.l     TextMark
        move.b    #" ",d1
NoEnd:  ext       d1
        sub.l     #32,d1
        asl.l     #2,d1
        lea       LetterTab(pc),a3
        add.l     (a3,d1.l),d0
        move.l    d0,BLTAPTH(a6)
        move.l    #Pic+48,d0
        add.l     d7,d0
        move.l    d0,BLTDPTH(a6)
        move.w    #2050,BLTSIZE(a6)
;
WaitLoop2:
        btst      #6,DMACONR(A6)
        bne.s     WaitLoop2
```

```
          add.l     #1664,d7
          add.l     #40,d6
          cmp.l     #200,d6
          bne.s     Nochmal
          add.l     #1,TextMark
NoLetter:           rts
;
;************************************
;
CopperChange:
          lea       Sinus(pc),a0
          add.l     SinusCount,a0
;
          move.w    #$68,d1
          add.b     $1(a0),d1
          move.b    d1,Cins1
          add.b     #$20,d1
          move.b    d1,Cins2
;
          add.l     #8,SinusCount
          cmp.l     #504,SinusCount
          bne.s     NoEnd2
          clr.l     SinusCount
NoEnd2:   rts
;************************************
even
Size:               dc.l   1664
LetterMark:         dc.w   0
TextMark:           dc.l   0
SinusCount:         dc.l   0
sprite:             dc.l   0
;
LetterTab:
dc.l 0,4,8,12,16,20,24,28,32,36
dc.l 6400,6404,6408,6412,6416,6420,6424,6428,6432,6436
dc.l 12800,12804,12808,12812,12816,12820,12824,12828,12832
```

```
dc.l 12836
dc.l 19200,19204,19208,19212,19216,19220,19224,19228,19232
dc.l 19236
dc.l 25600,25604,25608,25612,25616,25620,25624,25628,25632
dc.l 25636
dc.l 32000,32004,32008,32012,32016,32020,32024,32028,32032
dc.l 32036
;
Text:
dc.b " DIESE LAUFSCHRIFT IST 32X32 PIXEL GROSS UND
SPRINGT "
dc.b "AUF UND AB!!!          ",$ff
even
;
Sinus:    blk.b    504,0
Font:     blk.b    38464,0
Pic:      blk.b    8320,0  ;32*5*52 (Höhe*Tiefe*Breite)
```

5.2. OBJEKTE

Auf den nachfolgenden Seiten wollen wir uns einem Thema widmen, das oft in der Presse erwähnt wird. Die Blitterobjekte oder kurz Bobs genannt. Wenn man normalerweise ein Objekt darstellen und bewegen möchte, verwendet man dafür die Sprites, die der Amiga zur Verfügung stellt. Die Anwendung der Sprites ist ziemlich einfach, dafür hat hat dies einige Nachteile:

- Es stehen nur maximal acht Sprites zur Verfügung.
- Ein Sprite kann zwar beliebig hoch, aber nur maximal 16 Pixel breit sein.
- Ein Sprite besteht aus 4 Farben, wobei eine die Hintergrundfarbe und als solche durchsichtig ist. Es gibt einen Trick, mit dem man auch 16farbige Sprites darstellen kann, aber um ein 16farbiges Sprite darstellen zu können, muß man zwei normale Sprites zusammenlegen.

Wie Sie sehen, besitzen die Sprites einige Nachteile. Eine andere Möglichkeit sind die schon erwähnten Bobs.

Bobs können praktisch beliebig groß sein und eine beliebige Anzahl Farben besitzen. Die Größe und die Farbanzahl der Bobs ist abhänig von den Playfields, die verwendet werden. Bobs sind ein Bestandteil des Playfields, da sie in dieses hineinkopiert werden.

5.2.1. OBJEKTE NACH KOORDINATENTABELLE BEWEGEN

Wie man mit dem Blitter Objekte auf dem Bildschirm plaziert, haben wir schon in den ersten Kapiteln besprochen. Jedoch um ein Objekt zu bewegen, ist noch einiges zu berücksichtigen.

Der Blitter kann Daten nur an Wordgrenzen kopieren, das bedeutet, daß man nur alle 16 Punkte eine Grafik plazieren kann. Will man nun ein Bob bewegen, muß eine Plazierung an jeder Stelle des Bildschirmes möglich sein. Dazu muß das Objekt verschoben werden, auch Shiften genannt.

Wenn man ein Bob nach links verschiebt, werden die rechts liegenden Bits ins Bild geschoben. Sind diese Bits nicht gelöscht, dann entsteht ein unerwünschtes Bild. Daher muß rechts neben jedem Objekt, das verschoben werden soll, ein Abstand von 16 Pixeln freigelassen werden.

Sehen wir uns nun die Routine an, die die Daten für eine Kreisbewegung des 32 farbigen Bobs aus einer Tabelle holt und das Objekt an der richtigen Stelle plaziert.

```
move.l  #Pic,d0
lea     Coords(pc),a0
add.l   CoordCount,a0
```

Zuerst werden die Adressen der Bitplane, sowie der Koordinatentabelle in die Daten- bzw. Adressregister geladen.

```
; Y-Koordinate berechnen
;
        clr.l       d1
        move.w      $2(a0),d1
        mulu        #200,d1
        add.l       d1,d0
;
; X-Koordinate berechnen
;
        move.w      (a0),d1
        and.w       #$f,d1
        ror.w       #4,d1
        move.w      #%0000100111110000,d2
        add.w       d1,d2
        move.w      d2,BLTCON0(a6)
;
        clr.l       d1
        move.w      (a0),d1
        and.w       #$fff0,d1
        asr.w       #3,d1
        add.l       d1,d0
;
```

In diesem Abschnitt werden die Daten für die X- und die Y-Koordinate aus der Tabelle geholt und die dadurch entstehende Verschiebung in das BLTCON0-Register eingetragen.

```
        move.l      d0,SaveAdress
        move.l      d0,BLTDPTH(a6)
        move.w      #10243,BLTSIZE(a6)
```

Letztendlich wird der Blitter durch Beschreiben des BLTSIZE-Registers gestartet. Die BLITSIZE ergibt sich aus: (32*64)*5+2=10243

```
;           Ein Objekt nach Koordinatentabelle bewegen
;
;           Nachzuladende Files:
;
;           "KreisDat.tab"                auf "Coords"
;           "raw/Bob32x32_5pl-breit.con"  auf "Bob"
;
dmaconr     =           $02
bltddat     =           $00
bltcon0     =           $40
bltcon1     =           $42
bltafwm     =           $44
bltalwm     =           $46
bltcpth     =           $48
bltapth     =           $50
bltaptl     =           $52
bltdpth     =           $54
bltsize     =           $58
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
s:          bsr.s       opening
            bsr.L       initcolors
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$0ff00,d0
            bne.s       loop
;
            move.l      #$dff000,a6
            bsr.L       blit
```

```
        btst      #6,$bfe001
        bne.s     loop
;
;
end:    move.w    #$c000,$dff09a
        moveq.l   #0,d0
e:      rts
;
opening: move.w   #$4000,$dff09a
        move.l    #copperl,$dff084
        move.l    #pic,d0
        move      d0,pl1+6
        swap      d0
        move      d0,pl1+2
        swap      d0
        add.l     size(pc),d0
        move      d0,pl2+6
        swap      d0
        move      d0,pl2+2
        swap      d0
        add.l     size(pc),d0
        move      d0,pl3+6
        swap      d0
        move      d0,pl3+2
        swap      d0
        add.l     size(pc),d0
        move      d0,pl4+6
        swap      d0
        move      d0,pl4+2
        swap      d0
        add.l     size(pc),d0
        move      d0,pl5+6
        swap      d0
        move      d0,pl5+2
;
```

```
             move.l    #sprite,d0
             lea       sp1+2(pc),a0
             moveq.l   #7,d1
initsprites: swap      d0
             move      d0,(a0)
             add.l     #4,a0
             swap      d0
             move      d0,(a0)
             add.l     #4,a0
             dbf       d1,initsprites
             rts
;
initcolors:  lea       color(pc),a0
             lea       Bob+960(pc),a1
             moveq     #31,d0
             move      #$180,d1
copycolors:
             move      d1,(a0)+
             move      (a1)+,(a0)+
             addq      #2,d1
             dbf       d0,copycolors
;
             rts
;
copperl:     dc.w      $008e,$3090
             dc.w      $0090,$f8f0
             dc.w      $0092,$0038
             dc.w      $0094,$00d0
             dc.w      $0102,$0000
             dc.w      $0104,$0024
             dc.w      $0108,$00a0
             dc.w      $010a,$00a0
             dc.w      $0100,$5200
pl1:         dc.w      $00e0,$0000
             dc.w      $00e2,$0000
pl2:         dc.w      $00e4,$0000
```

```
        dc.w    $00e6,$0000
pl3:    dc.w    $00e8,$0000
        dc.w    $00ea,$0000
pl4:    dc.w    $00ec,$0000
        dc.w    $00ee,$0000
pl5:    dc.w    $00f0,$0000
        dc.w    $00f2,$0000
spl:    dc.w    $0120,$0000
        dc.w    $0122,$0000
        dc.w    $0124,$0000
        dc.w    $0126,$0000
        dc.w    $0128,$0000
        dc.w    $012a,$0000
        dc.w    $012c,$0000
        dc.w    $012e,$0000
        dc.w    $0130,$0000
        dc.w    $0132,$0000
        dc.w    $0134,$0000
        dc.w    $0136,$0000
        dc.w    $0138,$0000
        dc.w    $013a,$0000
        dc.w    $013c,$0000
        dc.w    $013e,$0000
color:  blk.l   32,0
        dc.w    $ffff,$fffe
;
;*******************************************
;
Blit:
;
; Löschen
        clr.w   BLTDDAT(a6)
        move.w  #34,BLTDMOD(a6)
        move.w  #%0000000100000000,BLTCON0(a6)
        move.l  SaveAdress,BLTDPTH(a6)
        move.w  #10243,BLTSIZE(a6)
```

```
Wait:   btst    #6,DMACONR(a6)
        bne.s   Wait
;
; Bob blitten
;
        clr.w   BLTCON1(a6)
        clr.w   BLTAMOD(a6)
        move.w  #34,BLTDMOD(a6)
        move.w  #$FFFF,BLTAFWM(a6)
        move.w  #$FFFF,BLTALWM(a6)
;
        move.l  #Bob,BLTAPTH(a6)

        move.l  #Pic,d0
        lea     Coords(pc),a0
        add.l   CoordCount,a0
;
; Y-Koordinate berechnen
;
        clr.l   d1
        move.w  $2(a0),d1
        mulu    #200,d1
        add.l   d1,d0
;
; X-Koordinate berechnen
;
        move.w  (a0),d1
        and.w   #$f,d1
        ror.w   #4,d1
        move.w  #%0000100111110000,d2
        add.w   d1,d2
        move.w  d2,BLTCON0(a6)
;
        clr.l   d1
        move.w  (a0),d1
        and.w   #$fff0,d1
```

```
        asr.w   #3,d1
        add.l   d1,d0
;
        move.l  d0,SaveAdress
        move.l  d0,BLTDPTH(a6)
        move.w  #10243,BLTSIZE(a6)

        add.l   #4,CoordCount
        cmp.l   #1008,CoordCount
        bne.L   Wait2
        clr.l   CoordCount

Wait2:  btst    #6,DMACONR(a6)
        bne.s   Wait2
        rts
;
;*******************************************
;
even
sprite: dc.l    0
size:   dc.l    40
SaveAdress:
        dc.l    pic
CoordCount:
        dc.l    0
Coords: blk.b   1008,0
Bob:    blk.b   1024,0
pic:    blk.b   40000,0
```

5.2.2. MEHRERE OBJEKTE GLEICHZEITIG BEWEGEN

Auf den vorhergehenden Seiten haben wir ein mehrfarbiges Objekt über den Bildschirm bewegt. Nun wollen wir noch einige Objekte mehr hinzufügen.

Dieses Programm unterscheidet sich vom letzten nur durch die Anordnung der Bewegungsroutine, die jetzt so umgebaut wurde, daß die Reihenfolge der Bobs in einer Tabelle festgelegt wird, die innerhalb einer Schleife jedem Objekt die richtigen Koordinaten zuordnet.

Dieses Programm soll demonstrieren, wie man mehrere farbige Objekte (mit 32 Farben, das entspricht 5 Bitplanes) mit derselben Geschwindigkeit in einer Richtung bewegt. Jedoch darf sich unter den Objekten kein Hintergrundbild befinden, da dieses sonst zerstört werden würde. Ebenso dürfen sich die Objekte nicht gegenseitig überlappen, da sie sich sonst gegenseitig kurzzeitig löschen würden.

Um dies zu demonstrieren, wurden die Abstände zwischen den 10 Objekten von uns so gewählt, daß sich die Bobs in den Kurven kurz überlagen. Da der Berührungspunkt nur sehr klein ist (immer nur in den Ecken), muß man genau hinsehen um die Kollisionen zu sehen.

```
;		Mehrere Objekte
;
; Mehrere GLEICHE Bobs, nach EINER Koordinatentablle,
; mit DERSELBEN Geschwindigkeit und Richtung und
; ohne mögliche Überlappung bewegen.
;
;		Nachzuladende Files:
;
;		"KreisDat.tab"			auf "Coords"
;		"raw/Bob32x32_5pl-breit.con"	auf "Bob"
;
```

```
dmaconr     =           $02
bltddat     =           $00
bltcon0     =           $40
bltcon1     =           $42
bltafwm     =           $44
bltalwm     =           $46
bltcpth     =           $48
bltapth     =           $50
bltaptl     =           $52
bltdpth     =           $54
bltsize     =           $58
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
s:          bsr.s       opening
            bsr.L       initcolors
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$0ff00,d0
            bne.s       loop
;
            move.l      #$dff000,a6
            bsr.L       blit
;
            btst        #6,$bfe001
            bne.s       loop
;
end:        move.w      #$c000,$dff09a
            moveq.l     #0,d0
e:          rts
```

```
opening:     move.w    #$4000,$dff09a
             move.l    #copperl,$dff084
             move.l    #pic,d0
             move      d0,pl1+6
             swap      d0
             move      d0,pl1+2
             swap      d0
             add.l     size(pc),d0
             move      d0,pl2+6
             swap      d0
             move      d0,pl2+2
             swap      d0
             add.l     size(pc),d0
             move      d0,pl3+6
             swap      d0
             move      d0,pl3+2
             swap      d0
             add.l     size(pc),d0
             move      d0,pl4+6
             swap      d0
             move      d0,pl4+2
             swap      d0
             add.l     size(pc),d0
             move      d0,pl5+6
             swap      d0
             move      d0,pl5+2
;
             move.l    #sprite,d0
             lea       sp1+2(pc),a0
             moveq.l   #7,d1
initsprites: swap      d0
             move      d0,(a0)
             add.l     #4,a0
             swap      d0
             move      d0,(a0)
             add.l     #4,a0
```

```
            dbf         d1,initsprites
            rts
;
initcolors: lea         color(pc),a0
            lea         Bob+960(pc),a1
            moveq       #31,d0
            move        #$180,d1
copycolors:
            move        d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
;
            rts
;
copperl:
            dc.w        $008e,$3090
            dc.w        $0090,$f8f0
            dc.w        $0092,$0038
            dc.w        $0094,$00d0
            dc.w        $0102,$0000
            dc.w        $0104,$0024
            dc.w        $0108,$00a0
            dc.w        $010a,$00a0
            dc.w        $0100,$5200
pl1:        dc.w        $00e0,$0000
            dc.w        $00e2,$0000
pl2:        dc.w        $00e4,$0000
            dc.w        $00e6,$0000
pl3:        dc.w        $00e8,$0000
            dc.w        $00ea,$0000
pl4:        dc.w        $00ec,$0000
            dc.w        $00ee,$0000
pl5:        dc.w        $00f0,$0000
            dc.w        $00f2,$0000
sp1:        dc.w        $0120,$0000
```

```
            dc.w      $0122,$0000
            dc.w      $0124,$0000
            dc.w      $0126,$0000
            dc.w      $0128,$0000
            dc.w      $012a,$0000
            dc.w      $012c,$0000
            dc.w      $012e,$0000
            dc.w      $0130,$0000
            dc.w      $0132,$0000
            dc.w      $0134,$0000
            dc.w      $0136,$0000
            dc.w      $0138,$0000
            dc.w      $013a,$0000
            dc.w      $013c,$0000
            dc.w      $013e,$0000
color:      blk.l     32,0
            dc.w      $ffff,$fffe
;
;*******************************************
;
Blit:
            lea       BobTable(pc),a5
            move.l    #0,d7
NoEnd:

; Löschen
;
            clr.w     BLTDDAT(a6)
            move.w    #34,BLTDMOD(a6)
            move.w    #%0000000100000000,BLTCON0(a6)
            move.l    $0(a5,d7.l),BLTDPTH(a6)
            move.w    #10243,BLTSIZE(a6)
;
; Bob blitten
;
            move.l    #Pic,d0
```

```
        lea         Coords(pc),a0
        add.l       $4(a5,d7.l),a0
;
; Y-Koordinate berechnen
;
        clr.l       d1
        move.w      $2(a0),d1
        mulu        #200,d1
        add.l       d1,d0
;
; X-Koordinate berechnen
;
        move.w      (a0),d1
        and.w       #$f,d1
        ror.w       #4,d1
        move.w      #%0000100111110000,d2
        add.w       d1,d2
;
        clr.l       d1
        move.w      (a0),d1
        and.w       #$fff0,d1
        asr.w       #3,d1
        add.l       d1,d0
        move.l      d0,(a5,d7.l)
;
Wait:   btst        #6,DMACONR(a6)
        bne.s       Wait
;
        move.w      d2,BLTCON0(a6)
        clr.w       BLTCON1(a6)
        clr.w       BLTAMOD(a6)
        move.w      #34,BLTDMOD(a6)
        move.w      #$FFFF,BLTAFWM(a6)
        move.w      #$FFFF,BLTALWM(a6)
        move.l      #Bob,BLTAPTH(a6)
        move.l      d0,BLTDPTH(a6)
```

```
            move.w     #10243,BLTSIZE(a6)
            add.l      #4,$4(a5,d7.l)
            cmp.l      #1008,$4(a5,d7.l)
            bne.L      Wait2
            clr.l      $4(a5,d7.l)
;
Wait2:      btst       #6,DMACONR(a6)
            bne.s      Wait2
;
            add.l      #8,d7
            cmp.l      #$FFFF,$0(a5,d7.l)
            bne.L      NoEnd
;
            rts
;
;*****************************************
;
even
sprite:     dc.l       0
size:       dc.l       40    ; Saveadress,CoordCount
BobTable:
            dc.l       pic,0
            dc.l       pic,80
            dc.l       pic,160
            dc.l       pic,240
            dc.l       pic,320
            dc.l       pic,400
            dc.l       pic,480
            dc.l       pic,560
            dc.l       pic,640
            dc.l       pic,720
            dc.l       $FFFF
;
Coords:     blk.b      1008,0
Bob:        blk.b      1024,0
pic:        blk.b      40000,0
```

5.2.3. 50 BOBS GLEICHZEITIG

In dem unten beschriebenen Listing versuchen wir die Anzahl der Bobs zu steigern, um möglichst viele Objekte darzustellen. Diesmal werden der Einfachheit halber nur Bobs mit einer Plane (2 Farben) dargestellt.

Wenn Sie das Listing näher betrachten, werden Sie feststellen, daß nur ein geringer Unterschied zu den beiden vorangegangenen besteht. Dieser Unterschied liegt wiederum in der Routine, die aus einer diesmal etwas längeren Tabelle den 50 Bobs die richtigen X- und Y-Werte zuordnet.

```
;           50 Bobs gleichzeitig
;           Nachzuladende Files:
;
;           "KreisDat.tab"                  auf "Coords"
;           "raw/Bob32x32_1-breit.raw"      auf "Bob"
;
dmaconr  =        $02
bltddat  =        $00
bltcon0  =        $40
bltcon1  =        $42
bltafwm  =        $44
bltalwm  =        $46
bltcpth  =        $48
bltbpth  =        $4c
bltapth  =        $50
bltaptl  =        $52
bltdpth  =        $54
bltsize  =        $58
bltcmod  =        $60
bltbmod  =        $62
bltamod  =        $64
bltdmod  =        $66
bltcdat  =        $70
```

```
bltbdat     =           $72
bltadat     =           $74
;
s:          bsr.s       opening
            bsr.L       initcolors
            bsr.L       MakeCoords
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$03000,d0
            bne.s       loop
;
            bsr.L       ChangePlane
            move.l      #$dff000,a6
            bsr.L       blit
;
            btst        #6,$bfe001
            bne.s       loop
;
end:        move.w      #$c000,$dff09a
            moveq.l     #0,d0
e:          rts
;
opening:    move.w      #$4000,$dff09a
            move.l      #copperl,$dff084
            move.l      #pic,d0
            move        d0,pl1+6
            swap        d0
            move        d0,pl1+2
;
            move.l      #sprite,d0
            lea         sp1+2(pc),a0
            moveq.l     #7,d1
initsprites: swap       d0
            move        d0,(a0)
            add.l       #4,a0
```

```
            swap        d0
            move        d0,(a0)
            add.l       #4,a0
            dbf         d1,initsprites
            rts
;
initcolors: lea         color(pc),a0
            lea         Colortable(pc),a1
            moveq       #1,d0
            move        #$180,d1
copycolors:
            move        d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
            rts
;
even
colortable: dc.w        $000,$fff
even
;
copperl:    dc.w        $008e,$3090
            dc.w        $0090,$f8f0
            dc.w        $0092,$0038
            dc.w        $0094,$00d0
            dc.w        $0102,$0000
            dc.w        $0104,$0024
            dc.w        $0108,$0000
            dc.w        $010a,$0000
            dc.w        $0100,$1200
pl1:        dc.w        $00e0,$0000
            dc.w        $00e2,$0000
sp1:        dc.w        $0120,$0000
            dc.w        $0122,$0000
            dc.w        $0124,$0000
            dc.w        $0126,$0000
```

```
        dc.w    $0128,$0000
        dc.w    $012a,$0000
        dc.w    $012c,$0000
        dc.w    $012e,$0000
        dc.w    $0130,$0000
        dc.w    $0132,$0000
        dc.w    $0134,$0000
        dc.w    $0136,$0000
        dc.w    $0138,$0000
        dc.w    $013a,$0000
        dc.w    $013c,$0000
        dc.w    $013e,$0000
color:  blk.l   2,0
        dc.w    $ffff,$fffe
;
;*******************************************
;
ChangePlane:
        eor.l   #10000,Puffer
;
        move.l  #pic,d0
        add.l   Puffer,d0
        move    d0,pl1+6
        swap    d0
        move    d0,pl1+2
;
        rts
;
;*******************************************
;
MakeCoords:
        lea     Coords(pc),a0
        lea     Coords2(pc),a1
        move.l  #0,d7
NoEnd4:
        move.l  #Pic,d0
```

```
; Y-Koordinate berechnen
;
          clr.l       d1
          move.w      $2(a0,d7.l),d1
          mulu        #40,d1
          add.l       d1,d0
;
; X-Koordinate berechnen
;
          move.w      (a0,d7.l),d1
          and.w       #$f,d1
          ror.w       #4,d1
          move.w      #%0000110111111100,d2
          add.w       d1,d2
;
          clr.l       d1
          move.w      (a0,d7.l),d1
          and.w       #$fff0,d1
          asr.w       #3,d1
          add.l       d1,d0
;
          move.l      d0,(a1)+
          move.w      d2,(a1)+
;
          add.l       #4,d7
          cmp.l       #1008,d7
          bne.s       NoEnd4
          rts
;*******************************************
Blit:
          eor.l       #4,puffer2
;
; Löschen
;
          move.w      #34,BLTDMOD(a6)
          move.w      #%0000000100000000,BLTCON0(a6)
```

```
;
            lea         BobTable(pc),a5
            move.l      puffer2,d1
NoEnd2:
            btst        #6,DMACONR(a6)
            bne.s       NoEnd2
;
            move.l      (a5,d1.l),BLTDPTH(a6)
            move.w      #2051,BLTSIZE(a6)
;
            add.l       #6,$8(a5)
            cmp.l       #1512,$8(a5)
            bne.L       NoZero
            clr.l       $8(a5)
NoZero:
            add.l       #12,a5
            cmp.b       #$FF,(a5)
            bne.s       NoEnd2
;
; Bob blitten
;
            clr.w       BLTCON1(a6)
            clr.w       BLTAMOD(a6)
            move.w      #34,BLTBMOD(a6)
            move.w      #34,BLTDMOD(a6)
            move.w      #$FFFF,BLTAFWM(a6)
            move.w      #$FFFF,BLTALWM(a6)
;
            lea         BobTable(pc),a5
NoEnd:      lea         Coords2(pc),a0
            add.l       $8(a5),a0
            move.l      (a0),d0
            add.l       puffer,d0
            move.l      d0,(a5,d1.l)
;
Wait:       btst        #6,DMACONR(a6)
```

```
            bne.s       Wait
;
            move.w      $4(a0),BLTCON0(a6)
            move.l      #Bob,BLTAPTH(a6)
            move.l      d0,BLTDPTH(a6)
            move.l      d0,BLTBPTH(a6)
            move.w      #2051,BLTSIZE(a6)
;
            add.l       #12,a5
            cmp.b       #$FF,(a5)
            bne.s       NoEnd
;
            rts
;*****************************************
even
Puffer:     dc.l        10000
Puffer2:    dc.l        0
;
; Saveadress,CoordCount
BobTable:
            dc.l        pic,pic,0
            dc.l        pic,pic,30
            dc.l        pic,pic,60
            dc.l        pic,pic,90
            dc.l        pic,pic,120
            dc.l        pic,pic,150
            dc.l        pic,pic,180
            dc.l        pic,pic,210
            dc.l        pic,pic,240
            dc.l        pic,pic,270
            dc.l        pic,pic,300
            dc.l        pic,pic,330
            dc.l        pic,pic,360
            dc.l        pic,pic,390
            dc.l        pic,pic,420
            dc.l        pic,pic,450
```

```
dc.l        pic,pic,480
dc.l        pic,pic,510
dc.l        pic,pic,540
dc.l        pic,pic,570
dc.l        pic,pic,600
dc.l        pic,pic,630
dc.l        pic,pic,660
dc.l        pic,pic,690
dc.l        pic,pic,720
dc.l        pic,pic,750
dc.l        pic,pic,780
dc.l        pic,pic,810
dc.l        pic,pic,840
dc.l        pic,pic,870
dc.l        pic,pic,900
dc.l        pic,pic,930
dc.l        pic,pic,960
dc.l        pic,pic,990
dc.l        pic,pic,1020
dc.l        pic,pic,1050
dc.l        pic,pic,1080
dc.l        pic,pic,1110
dc.l        pic,pic,1140
dc.l        pic,pic,1170
dc.l        pic,pic,1200
dc.l        pic,pic,1230
dc.l        pic,pic,1260
dc.l        pic,pic,1290
dc.l        pic,pic,1320
dc.l        pic,pic,1350
dc.l        pic,pic,1380
dc.l        pic,pic,1410
dc.l        pic,pic,1440
dc.l        pic,pic,1470
dc.l        pic,pic,1500
```

```
be:         dc.w     $FFFF
;
sprite:     dc.l     0
Coords:     blk.b    1008,0
Coords2:    blk.b    1512,0
Bob:        blk.b    192,0
pic:        blk.b    20000,0
```

Abschließend zu diesem Listing sei gesagt, daß es sehr wohl möglich ist, ein Vielfaches der dargestellten Bobs auf den Bildschirm zu bringen. Die Grenze nach oben ist noch lange nicht erreicht. Aber für unsere Demonstrationszwecke reichen 50 Bobs.

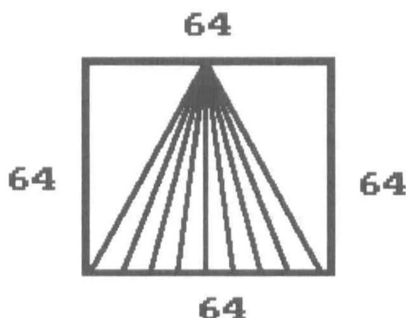
Denn wenn noch etwas mehr als 50 Bobs dargestellt werden würden, könnte das Auge keine Bewegung mehr feststellen, da zuviele Objekte auf der Kreisbahn bewegt werden würden. Aus diesem Grund haben wir uns mit 50 Bobs begnügt.

5.2.4. BOBS ZOOMEN

Den Abschluß des Kapitels Objekte bildet das Thema "Zoomen". Die Aufgabe ist, ein Bob mit der ursprünglichen Größe von 16 Pixel Höhe und 16 Pixel Breite stufenlos zu vergrößern und zu verkleinern. Das ganze soll natürlich in Echtzeit geschehen.

Wenn Sie das Kapitel "Zoom-Laufschrift" aufmerksam durchgearbeitet haben, wird Ihnen auch dieses Kapitel keine Probleme bereiten, denn die Technik des Zoomen von Bobs ist dieselbe, die wir bei der Zoom-Laufschrift angewandt haben.

Die Zoom-Routine wird wieder in zwei Teile unterteilt, die horizontale und die vertikale Vergrößerung. Um das Bob horizontal zu vergrößern, wird genau wie bei der Zoom-Laufschrift eine Maske verwendet, die einen stufenlosen Übergang garantiert.



Bei der Zoom-Laufschrift wurde jeweils eine Zeile dieser Maske mit Hilfe der Modulo-Werte über den gesamten Bildschirm dargestellt. Hier können wir die Copper-Modulo-Werte nicht verwenden, da unser Bob nicht den gesamten Bildschirm ausfüllt. Daher wird jede Zeile mit dem Blitter kopiert. Damit man aber gleiche Zeilen nicht doppelt kopieren muß, werden diese mit negativen Blitter-Modulo dargestellt.

Bei der vertikalen Vergrößerung müssen einige Zeilen doppelt dargestellt, bei der Verkleinerung hingegen einige Zeilen weggelassen werden. Diese Anordnung wurde in der Tabelle "Boz_3pl.tab" abgespeichert, die vor dem Programmstart von der Beispieldiskette nachgeladen werden muß.

Die Vergrößerung, bzw. Verkleinerung wäre soweit ganz einfach. Es fehlt nur noch die richtige Farbgebung, damit auch jedes Pixel bei der nächsten Vergrößerungs- oder Verkleinerungsphase die richtige Farbe zugeordnet bekommt. Damit dies stimmt, wurde eine zweite Maske und eine zweite Tabelle angefertigt, die analog zu der Vergrößerung jedem Pixel die richtige Farbe zuordnet.

```
;       Bob zoomen
;       Nachzuladende Files:
;       "Boz_3pl.tab"           auf "tab"
;       "Boz_3pl.tab2"          auf "tab2"
;       "ColorMask.con"         auf "ColorMask"
```

```
;           "BobZoomMask.con"           auf "msk"
;           "Bob_16_3pl.con"            auf "bob"
;
dmaconr     =           $02
bltddat     =           $00
bltcon0     =           $40
bltcon1     =           $42
bltafwm     =           $44
bltalwm     =           $46
bltcpth     =           $48
bltbpth     =           $4c
bltapth     =           $50
bltaptl     =           $52
bltdpth     =           $54
bltsize     =           $58
bltcmod     =           $60
bltbmod     =           $62
bltamod     =           $64
bltdmod     =           $66
bltcdat     =           $70
bltbdat     =           $72
bltadat     =           $74
;
s:          bsr.s       opening
            bsr.L       initcolors
loop:       move.l      $dff004,d0
            and.l       #$1ff00,d0
            cmp.l       #$07000,d0
            bne.s       loop
;
            bsr.L       ChangePlane
            move.l      #$dff000,a6
            bsr.L       Blit
;
            btst        #6,$bfe001
            bne.s       loop
```

```
end:        move.w   #$c000,$dff09a
            moveq.l  #0,d0
e:          rts
;
opening:    move.w   #$4000,$dff09a
            move.l   #copperl,$dff084
            move.l   #Pic,d0
            move     d0,pl1+6
            swap     d0
            move     d0,pl1+2
            swap     d0
            add.l    Size(pc),d0
            move     d0,pl2+6
            swap     d0
            move     d0,pl2+2
            swap     d0
            add.l    Size(pc),d0
            move     d0,pl3+6
            swap     d0
            move     d0,pl3+2
            swap     d0
;
            move.l   #sprite,d0
            lea      sp1+2(pc),a0
            moveq.l  #7,d1
initsprites: swap    d0
            move     d0,(a0)
            add.l    #4,a0
            swap     d0
            move     d0,(a0)
            add.l    #4,a0
            dbf      d1,initsprites
            rts
;
initcolors: lea      color(pc),a0
            lea      Bob+96(pc),a1
```

```
            moveq       #7,d0
            move        #$180,d1
copycolors: move        d1,(a0)+
            move        (a1)+,(a0)+
            addq        #2,d1
            dbf         d0,copycolors
;
            rts
;
;***************************************
;
Blit:       btst        #6,DMACONR(a6)
            bne.s       Blit
;
; Löschen
;
            move.w      #%0000000100000000,BLTCON0(a6)
            clr.w       BLTCON1(a6)
            move.w      #32,BLTDMOD(a6)
            move.l      #Pic+16,d0
            add.l       Puffer,d0
            move.l      d0,BLTDPTH(a6)
            move.w      #12292,BLTSIZE(a6)
;
Wait:       btst        #6,DMACONR(a6)
            bne.s       Wait
;
; Blitten
            clr.w       BLTCON1(a6)
            move.w      #-4,BLTAMOD(a6)
            move.w      #4,BLTCMOD(a6)
            move.w      #36,BLTBMOD(a6)
            move.w      #36,BLTDMOD(a6)
            move.w      #$FFFF,BLTAFWM(a6)
            move.w      #$FFFF,BLTALWM(a6)
            move.w      #%0000111111101100,$dff040
```

```
          move.l    #$dff048,a4
          move.l    #$dff04c,a5
          move.l    #$dff050,a6
;
          clr.l     d1
          move.w    #15,d7
          move.l    #Msk,d6
          add.l     ZoomCount(pc),d6
;
          move.l    #Pic+16,d4
          add.l     Puffer,d4
;
          lea       ColorMask(pc),a3
NextRow:  move.l    #Bob,a0
          clr.l     d5
          move.l    #15,d3
;
          move.l    #Tab,a1
          move.l    ZoomCount(pc),d2
          asl.l     #3,d2
          add.l     d2,a1
;
          move.l    #Tab2,a2
          move.l    ZoomCount(pc),d2
          asr.l     #2,d2
          add.l     d2,a2
;
          cmp.w     #7,d7
          bne.s     NextLine
          addq      #4,d4
          addq      #4,d6
NextLine: move.l    d6,(a6)             ; Maske in Quelle A
          clr.l     d0
          move.w    (a2),d0
          add.l     d4,d0
          add.l     d5,d0
```

```
          move.l    d0,(a5)             ; Plane in Quelle B
          move.l    d0,$4(a6)           ; und Ziel D
;
          move.w    (a1)+,d1
          beq.s     NoRow
          add.w     (a1),d5
;
          clr.l     d2
          move.w    (a0),d0
          btst      d7,d0
          beq.s     NoPixel1
          addq      #1,d2
NoPixel1: move.w    $20(a0),d0
          btst      d7,d0
          beq.s     NoPixel2
          addq      #2,d2
NoPixel2: move.w    $40(a0),d0
          btst      d7,d0
          beq.s     NoPixel3
          addq      #4,d2
NoPixel3: cmp.b     #0,d2
          beq.s     NoPixel
;
          mulu      #384,d2
          add.l     a3,d2
          move.l    d2,(a4)             ;ColorMask in Quelle C
;
          move.w    d1,$8(a6)           ; BLTSIZE
NoRow:
NoPixel:  addq      #2,a1
          addq      #2,a0
          dbf       d3,NextLine
;
          add.l     #512,d6
          dbf       d7,NextRow
;
```

```
        add.l     #8,ZoomCount
        cmp.l     #512,ZoomCount
        bne.s     NoZero
        clr.l     ZoomCount
NoZero: rts
;
;*****************************************
;
ChangePlane:
        move.l    #Pic,d0
        add.l     Puffer,d0
        move      d0,pl1+6
        swap      d0
        move      d0,pl1+2
        swap      d0
        add.l     Size(pc),d0
        move      d0,pl2+6
        swap      d0
        move      d0,pl2+2
        swap      d0
        add.l     Size(pc),d0
        move      d0,pl3+6
        swap      d0
        move      d0,pl3+2
        swap      d0
;
        eor.l     #24000,Puffer
;
        rts
;
;*****************************************
;
copperl: dc.w     $008e,$3090
        dc.w      $0090,$f8f0
        dc.w      $0092,$0038
        dc.w      $0094,$00d0
```

```
        dc.w     $0102,$0000
        dc.w     $0104,$0024
        dc.w     $0108,$0050
        dc.w     $010a,$0050
        dc.w     $0100,$3000
pl1:    dc.w     $00e0,$0000
        dc.w     $00e2,$0000
pl2:    dc.w     $00e4,$0000
        dc.w     $00e6,$0000
pl3:    dc.w     $00e8,$0000
        dc.w     $00ea,$0000
sp1:    dc.w     $0120,$0000
        dc.w     $0122,$0000
        dc.w     $0124,$0000
        dc.w     $0126,$0000
        dc.w     $0128,$0000
        dc.w     $012a,$0000
        dc.w     $012c,$0000
        dc.w     $012e,$0000
        dc.w     $0130,$0000
        dc.w     $0132,$0000
        dc.w     $0134,$0000
        dc.w     $0136,$0000
        dc.w     $0138,$0000
        dc.w     $013a,$0000
        dc.w     $013c,$0000
        dc.w     $013e,$0000
color:  blk.l    8,0
        dc.w     $ffff,$fffe
;
even
sprite: dc.l     0
Size:   dc.l     40
Puffer: dc.l     24000
ZoomCount:
        dc.l     80
```

```
Tab:        blk.b       4096,0
Tab2:       blk.b       128,0
Bob:        blk.b       112,0
ColorMask:
            blk.b       3072,0
Msk:        blk.b       8192,0                  ; 16*512
Pic:        blk.b       48000,0
;
```

6. SONSTIGES

An dieser Stelle wollen wir uns jenen Dingen widmen, die man nicht so recht in eines der vorhergehenden Kapitel einordnen kann.

Wir programmieren noch einen kleinen Effekt, der für einige Programmierer sicherlich interessant ist. Dabei handelt es sich um einen Sternen-Himmel, der mit möglichst einfachen Mitteln die optimale Wirkung erzielt.

Abschließend werden einige Anregungen gegeben, wie Sie das Gelernte aus Band 1 und Band 2 von "Spiele selber programmieren" auch praktisch anwenden oder kombinieren können.

6.1. JOYSTICK-ABFRAGE

Ein wesentlicher Bestandteil eines Spieles ist wohl die Steuerung, denn nichts regt einen Spieler mehr auf, als eine schlechte Umsetzung der Bewegungen. Daher sollte man auf diesen Punkt ebenfalls sehr achten.

Es gibt mehrere Möglichkeiten, den Joystickport auszulesen. Wir verwenden eine der kürzesten und schnellsten Methoden. Wir lesen das Joystick-Register $dff00c (JOY1DAT) in ein Datenregister ein und überprüfen den Inhalt. Wird nun der Joystick in eine Richtung bewegt, so stehen in d1 folgende Werte:

0003	rechts
0300	links
0100	oben
0001	unten
0200	links + oben
0002	rechts + unten
0103	rechts + oben
0301	links + unten

Wir müssen also lediglich mit cmp das Datenregister auf einen dieser Werte überprüfen, um bei erfolgreichem Vergleich in die gewünschte Unterroutine zu verzweigen. Für Port 0 (Mausport) ist folgendes Register zu verwenden:

JOY0DAT $dff00a

Die fertige Joystickabfrage für unser Spiel (das bereits in Band 1 besprochen wurde) benötigt nur vier Bewegungsrichtungen. Wurde der Hebel gedrückt, verzweigt das Programm in die jeweilige Unterroutine und addiert bzw. subtrahiert 1 von der Spriteadresse. Ebenfalls wird verglichen, ob das Sprite schon eine Randgrenze erreicht hat. Wenn ja, wird nicht addiert/subtrahiert, sondern gleich zum Ende der Routine verzweigt.

```
; Die komplette Joystick-Abfrage
;
joy:        cmp.b       #$01,fin
beq.s       joyout
            clr         $dff036
            move        $dff00c,d1
            cmp         #$0300,d1
            beq.s       links
            cmp.b       #$03,d1
            beq.s       rechts
            cmp         #$0100,d1
            beq.s       oben
            cmp.b       #$01,d1
            beq.s       unten
joyout:     rts
links:      move.l      spritebuffer1,d0
            swap        d0
            cmp.b       #$40,d0
            bls.s       joyout
            sub         #$0001,spritebuffer1
            rts
```

```
rechts:     move.l      spritebuffer1,d0
            swap        d0
            cmp.b       #$d8,d0
            bhs.s       joyout
            add         #$0001,spritebuffer1
            rts
oben:       move.l      spritebuffer1,d0
            swap        d0
            clr.b       d0
            cmp         #$3000,d0
            bls.s       joyout
            sub.l       #$01000100,spritebuffer1
            rts
unten:      move.l      spritebuffer1,d0
            swap        d0
            clr.b       d0
            cmp         #$ef00,d0
            bhs.s       joyout
            add.l       #$01000100,spritebuffer1
            rts
;
```

Ein weiterer Punkt ist das Auslesen der Feuerknöpfe. Die Abfrage des Joystickbuttons ist genau die gleiche wie die der Maus:

```
button:     btst        #6,$bfe001
            bne.s       button
```

Die Abfrage für Port 1 liegt nur ein Bit weiter, nämlich:

```
button:     btst        #7,$bfe001
            bne.s       button
```

6.2. TASTATUR-ABFRAGE

Fast ebenso wichtig wie die Joystick-Abfrage ist die Auswertung der Tastatur. In dem in Band 1 vorgestellten Spiel kommt diese zwar nicht zur Anwendung, aber trotzdem ist dieses Kapitel interessant, da die Tastatur sehr häufig verwendet wird.

Der Amiga besitzt ein intelligentes Keyboard mit einem eigenen Prozessor, der automatisch die Auswertung übernimmt. Wir müssen lediglich den Puffer, in welchem die Daten gespeichert werden, auslesen.

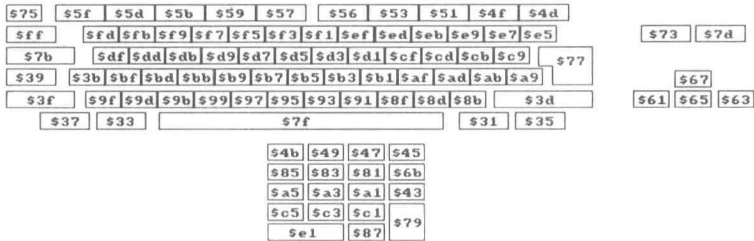
Zuerst rettet man den Inhalt des Registers $bfec01, in dem sich die zuletzt gedrückte Taste befindet und vergleicht anschließend diesen mit den Tastatur-Codes.

```
loop:       move.b      $bfec01,d0
            cmp.b       #$41,d0
            bne.s       loop
            rts
```

Dieses kurze Programm wartet solange, bis die Help-Taste gedrückt wurde. Die Codes für die verschiedenen Tasten entnehmen Sie bitte der untenstehenden Abbildung. Diese Abbildung entspricht der normalen Amiga 500 Tastatur. Es wurden aber auch schon andere Tastatur-Arten ausgeliefert z.B. beim Amiga 1000. Diese Tastaturen unterscheiden sich in der Anordnung der Tasten sowie in der Anzahl derselben. Es gibt Tastaturen, die um einige Tasten erweitert wurden. Wenn man obiges Programm so verändert, daß es den Wert der gerade gedrückten Zahl liefert, so kann man auch die Codes der nicht angeführten Tasten leicht ermitteln.

Wie Sie vielleicht bemerkt haben, entsprechen die angegebenen Codes nicht den ASCII-Codes. Will man diese erhalten, muß man eine Tabelle anlegen, die jedem Code das passende ASCII-Zeichen zuordnet.

Bild 8: Die Tastatur



6.3. STERNENHIMMEL

Ein ebenfalls sehr häufig verwendeter Effekt ist das Sternenscrolling. Wir wollen an dieser Stelle versuchen, diesen Effekt nachzuprogrammieren. Folgendes Ziel wollen wir uns dabei setzen:

- 80 Sterne sollen gleichzeitig über den Bildschirm bewegt werden.
- Sie sollen in unterschiedlichen Geschwindigkeiten aus der Bildschirmmitte dem Betrachter entgegenfliegen.

Diese Aufgabenstellung hört sich schwieriger an, als sie in Wirklichkeit ist. Beginnen wir einmal mit dem Einfachsten, dem Erstellen eines Bildschirms mit nur einer Plane. Da unsere Sterne einfarbig sind, genügt uns eine Bitplane.

Als nächstes legen wir eine Tabelle mit den Startkoordinaten für X und Y an. Diese Koordinaten geben die Ausgangspositionen unserer Sterne an:

```
skorsx:   dc.w 70,80,90,100,110,120,130,140,150,160
          dc.w 75,89,34,45,145,123,23,43,100,139
          ....

skorsy:   dc.w 55,60,65,70,75,80,85,90,95,100
          dc.w 56,76,64,45,67,98,65,54,76,86
          ....
```

Diese Positionen sind willkürlich im Koordinatensystem gewählt und können geändert werden. In jeder Zeile sind 10 Werte eingetragen, das bedeutet, daß noch 60 weitere fehlen, die wir der Einfachheit halber mit denselben Werten belegt haben.

Es folgen zwei weitere Tabellen, die die Richtung und die Geschwindigkeit der Sterne innerhalb eines Quadranten festlegen.

```
speed1:     dc.w 1,2,3,4,5,6,2,4,5,6
            dc.w 6,4,0,1,3,6,1,7,3,3
            dc.w -1,-2,-3,-4,-5,-6,-2,-4,-5,-6
            dc.w -6,-4,-0,-1,-3,-6,-1,-7,-3,-3
            dc.w -1,-2,-3,-4,-5,-6,-2,-4,-5,-6
            dc.w -6,-4,-0,-1,-3,-6,-1,-7,-3,-3
            dc.w 1,2,3,4,5,6,2,4,5,6
            dc.w 6,4,0,1,3,6,1,7,3,3

speed2:     dc.w 1,1,1,1,1,1,1,1,1,1
            dc.w 1,1,1,1,1,1,0,0,1,1
            dc.w 1,1,1,1,1,1,1,1,1,1
            dc.w 1,1,1,1,1,1,0,0,1,1
            dc.w -1,-1,-1,-1,-1,-1,-1,-1,-1,-1
            dc.w -1,-1,-1,-1,-1,-1,0,0,-1,-1
            dc.w -1,-1,-1,-1,-1,-1,-1,-1,-1,-1
            dc.w -1,-1,-1,-1,-1,-1,0,0,-1,-1
```

Widmen wir uns als nächstes der eigentlichen Bewegungsroutine. Erst retten wir sämtliche Register, indem wir diese auf dem Stack ablegen.

```
movestars:
            movem.l d0-d7/a0-a6,-(a7)
```

Der nächste Schritt ist das Einlesen der Koordinaten, sowie der Geschwindigkeitsparameter. Danach wird die Anzahl der Sterne in d7 übergeben.

```
        lea         skorsx(pc),a0
        lea         skorsy(pc),a1
        lea         speed1(pc),a2
        lea         speed2(pc),a3
        move.l      #79,d7
```

Nachdem wir alle Parameter in den einzelnen Registern übergeben haben, können wir in der nächsten Routine mit der Berechnung der richtigen Bildschirmposition beginnen. Die folgenden Zeilen erledigen diese Aufgabe für uns. Zuerst werden die Koordinaten mit denen der äußersten Ränder verglichen. Ist ein Stern über diese Koordinate bewegt worden, so wird er wieder auf die Ausgangsposition (x=160, y=128) zurückgesetzt. Dieser Vergleich muß natürlich für alle vier Bildschirmseiten gemacht werden.

```
starloop:   move    (a0),d0
            move    (a1),d1
            move    (a2)+,d2
            move    (a3)+,d3
            move    d0,d4
            move    d1,d5
            sub     d2,d0
            sub     d3,d1
            cmp     #370,d0
            ble.s   nox
            move    #160,d0
            move    #128,d1
            bra.s   noy
nox:        cmp     #9,d0
            bge.s   nomix
            move    #160,d0
            move    #128,d1
            bra.s   nomiy
nomix:      cmp     #270,d1
            ble.s   noy
            move    #160,d0
```

```
        move    #128,d1
        bra.s   nomiy
noy:    cmp     #9,d1
        bge.s   nomiy
        move    #160,d0
        move    #128,d1
nomiy:  move.w  d0,(a0)
        move.w  d1,(a1)+
```

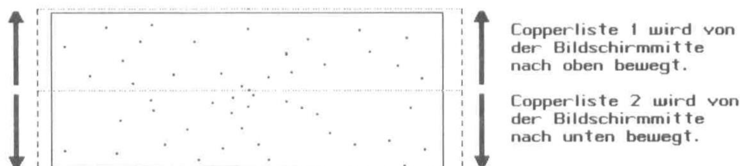
Der letzte Schritt ist das Berechnen der Bildschirmposition, sowie das Löschen des zuletzt bewegten Sterns und das Setzen des neuen Punktes. Somit wäre unsere Routine für einen Stern komplett. Damit der Computer aber 80 Sterne bewegt, müssen wir am Ende mit einer dbf-Schleife den Vorgang weitere 79 Male wiederholen. Abschließend werden die geretteten Register wieder zurückgeschrieben.

```
        mulu    #48,d5
        lsr.w   #3,d4
        lea     pic(pc),a5
        add     d4,a5
        add     d5,a5
        clr.b   (a5)
;
        lea     pic(pc),a5
        lsr.w   #3,d0
        mulu    #48,d1
        add     d0,a5
        add     d1,a5
;
        move.w  (a0)+,d0
        and.w   #$7,d0
        move.l  #128,d1
        lsr.l   d0,d1
        move.b  d1,(a5)
;
```

```
        dbf         d7,starloop
        movem.l     (a7)+,d0-d7/a0-a6
        rts
```

Die Stern-Routine hätten wir somit fertiggestellt. 80 Punkte fliegen von der Bildschirmmitte dem Betrachter quasi entgegen. Da sie in verschiedenen Geschwindigkeiten bewegt werden, wirkt unser kleines Weltall fast realistisch. Um diesen 3D-Effekt noch zu verstärken, werden wir den Sternen verschiedene Farben geben (Helligkeitsstufen), damit der Eindruck erweckt wird, als ob einige weiter im Hintergrund bewegt werden. Diesen Effekt werden wir ohne Zuhilfenahme einer weiteren Bitplane programmieren, sondern wir werden dazu den Copper verwenden. Wie schon aus der unten stehenden Skizze ersichtlich, verwenden wir zwei Farb-Copperlisten, wobei eine von oben nach unten und die andere von unten nach oben gescrollt wird. Zuerst müssen diese Listen initialisiert werden.

Bild 9: Sternenhimmel




```
;
initcopper: lea     cins(pc),a0
            move    #$0001,d0
            move    #159,d1
coppercopy:
            move    d0,(a0)+
            move    #$fffe,(a0)+
            move    #$0182,(a0)+
            add.w   #1,d2
            move    d2,(a0)+
            add     #$0100,d0
            dbf     d1,coppercopy
```

Die oben stehende Copperliste erstellt pro Bildschirmzeile einen Wait- und einen Farb-Move-Befehl. Die zweite Copperliste wird analog dazu installiert. Sehen wir uns als nächstes jene Routien an, die die Farben der Copperliste scrollt:

```
;
scrollcopper1:
            lea         cins+6(pc),a0
            move.l      colorptr1(pc),a1
            cmp.w       #$ffff,(a1)
            bne.s       dascroll
            move.l      #color1,colorptr1
            move.l      colorptr1(pc),a1
dascroll:   move.w      #159,d0
scrollen:   move.w      (a1)+,(a0)
            cmp.w       #$ffff,(a1)
            bne.s       dascrollen
            lea         color1(pc),a1
dascrollen: add.l       #8,a0
            dbf         d0,scrollen
            add.l       #2,colorptr1
            rts
```

Als Abschluß finden Sie das komplette Listing, das 80 Sterne darstellt, in verschiedenen Geschwindigkeiten auf dem Bildschirm bewegt, sowie durch den Trick mit zwei Copperlisten den Effekt der verschiedenen Helligkeitsstufen erzeugt.

```
;       Sternenhimmel
;
s:          bsr.s       opening
            bsr.l       initcopper
;
loop:       move.l      $dff004,d0
            and.l       #$fff00,d0
            cmp.l       #$00003000,d0
```

```
        bne.s   loop
        bsr.l   movestars
        bsr     scrollcopper1
        bsr     scrollcopper2
        btst    #6,$bfe001
        bne.s   loop
;
end:    move.w  #$c000,$dff09a
e:      rts
;
opening: move.w #$4000,$dff09a
        move.l  #copperl,$dff084
        move.w  #$20,$dff096
        move.l  #pic,d0
        move    d0,pl1+6
        swap    d0
        move    d0,pl1+2
        rts
;
copperl:
        dc.w    $008e,$2071
        dc.w    $0090,$20d4
        dc.w    $0092,$0038
        dc.w    $0094,$00d0
        dc.w    $0102,$0000
        dc.w    $0104,$0024
        dc.w    $0108,$0008
        dc.w    $010a,$0008
        dc.w    $0100,$1200
pl1:    dc.w    $00e0,$0000
        dc.w    $00e2,$0000
        dc.w    $0180,$0000
cins:   blk.w   640,0
cins2:  blk.w   640,0
        dc.w    $ffff,$fffe
;
```

```
initcopper: lea         cins(pc),a0
            move        #$0001,d0
            move        #159,d1
coppercopy:
            move        d0,(a0)+
            move        #$fffe,(a0)+
            move        #$0182,(a0)+
            add.w       #1,d2
            move        d2,(a0)+
            add         #$0100,d0
            dbf         d1,coppercopy
            lea         cins2(pc),a0
            move        #$a001,d0
            move        #159,d1
coppercopy1:
            move        d0,(a0)+
            move        #$fffe,(a0)+
            move        #$0182,(a0)+
            add.w       #1,d2
            move        d2,(a0)+
            add         #$0100,d0
            dbf         d1,coppercopy1
            rts
;
scrollcopper1:
            lea         cins+6(pc),a0
            move.l      colorptr1(pc),a1
            cmp.w       #$ffff,(a1)
            bne.s       dascroll
            move.l      #color1,colorptr1
            move.l      colorptr1(pc),a1
dascroll:   move.w #159,d0
scrollen:   move.w (a1)+,(a0)
            cmp.w       #$ffff,(a1)
            bne.s       dascrollen
            lea         color1(pc),a1
```

```
dascrollen:add.l        #8,a0
           dbf          d0,scrollen
           add.l        #2,colorptr1
           rts
;
scrollcopper2:
           lea          cins2+1278(pc),a0
           move.l       colorptr2(pc),a1
           cmp.w        #$ffff,(a1)
           bne.s        dascroll2
           move.l       #color2,colorptr2
           move.l       colorptr2(pc),a1
dascroll2: move.w #159,d0
scrollen2: move.w (a1)+,(a0)
           cmp.w        #$ffff,(a1)
           bne.s        dascrollen2
           lea          color2(pc),a1
dascrollen2:
           sub.l        #8,a0
           dbf          d0,scrollen2
           add.l        #2,colorptr2
           rts
;
colorptr1: dc.l  color1
color1:    dc.w $fff,$eee,$ddd,$ccc,$bbb,$aaa,$999,$888
           dc.w $777,$666,$555,$444,$333,$222,$111,$000
           dc.w $000,$111,$222,$333,$444,$555,$666,$777
           dc.w $888,$999,$aaa,$bbb,$ccc,$ddd,$eee,$fff
           dc.w $ffff
;
colorptr2: dc.l  color2
color2:    dc.w $fff,$eee,$ddd,$ccc,$bbb,$aaa,$999,$888
           dc.w $777,$666,$555,$444,$333,$222,$111,$000
           dc.w $000,$111,$222,$333,$444,$555,$666,$777
           dc.w $888,$999,$aaa,$bbb,$ccc,$ddd,$eee,$fff
           dc.w $ffff
```

```
movestars: movem.l d0-d7/a0-a6,-(a7)
           lea      skorsx(pc),a0
           lea      skorsy(pc),a1
           lea      speed1(pc),a2
           lea      speed2(pc),a3
           move.l   #79,d7
starloop:  move     (a0),d0
           move     (a1),d1
           move     (a2)+,d2
           move     (a3)+,d3
           move     d0,d4
           move     d1,d5
           sub      d2,d0
           sub      d3,d1
           cmp      #370,d0
           ble.s    nox
           move     #160,d0
           move     #128,d1
           bra.s    noy
nox:       cmp      #9,d0
           bge.s    nomix
           move     #160,d0
           move     #128,d1
           bra.s    nomiy
nomix:     cmp      #270,d1
           ble.s    noy
           move     #160,d0
           move     #128,d1
           bra.s    nomiy
noy:       cmp      #9,d1
           bge.s    nomiy
           move     #160,d0
           move     #128,d1
nomiy:     move.w d0,(a0)
           move.w d1,(a1)+
;
```

```
        mulu        #48,d5
        lsr.w       #3,d4
        lea         pic(pc),a5
        add         d4,a5
        add         d5,a5
        clr.b       (a5)
;
        lea         pic(pc),a5
        lsr.w       #3,d0
        mulu        #48,d1
        add         d0,a5
        add         d1,a5
;
        move.w (a0)+,d0
        and.w       #$7,d0
        move.l      #128,d1
        lsr.l       d0,d1
        move.b      d1,(a5)
;
        dbf         d7,starloop

        movem.l (a7)+,d0-d7/a0-a6
        rts
;
speed1: dc.w 1,2,3,4,5,6,2,4,5,6
        dc.w 6,4,0,1,3,6,1,7,3,3

        dc.w -1,-2,-3,-4,-5,-6,-2,-4,-5,-6
        dc.w -6,-4,-0,-1,-3,-6,-1,-7,-3,-3

        dc.w -1,-2,-3,-4,-5,-6,-2,-4,-5,-6
        dc.w -6,-4,-0,-1,-3,-6,-1,-7,-3,-3

        dc.w 1,2,3,4,5,6,2,4,5,6
        dc.w 6,4,0,1,3,6,1,7,3,3
```

```
speed2:   dc.w 1,1,1,1,1,1,1,1,1,1
          dc.w 1,1,1,1,1,1,0,0,1,1
          dc.w 1,1,1,1,1,1,1,1,1,1
          dc.w 1,1,1,1,1,1,0,0,1,1

          dc.w -1,-1,-1,-1,-1,-1,-1,-1,-1,-1
          dc.w -1,-1,-1,-1,-1,-1,0,0,-1,-1

          dc.w -1,-1,-1,-1,-1,-1,-1,-1,-1,-1
          dc.w -1,-1,-1,-1,-1,-1,0,0,-1,-1

skorsx:   dc.w 70,80,90,100,110,120,130,140,150,160
          dc.w 75,89,34,45,145,123,23,43,100,139

          dc.w 70,80,90,100,110,120,130,140,150,160
          dc.w 75,89,34,45,145,123,23,43,100,139

          dc.w 70,80,90,100,110,120,130,140,150,160
          dc.w 75,89,34,45,145,123,23,43,100,139

          dc.w 70,80,90,100,110,120,130,140,150,160
          dc.w 75,89,34,45,145,123,23,43,100,139

skorsy:   dc.w 55,60,65,70,75,80,85,90,95,100
          dc.w 56,76,64,45,67,98,65,54,76,86

          dc.w 55,60,65,70,75,80,85,90,95,100
          dc.w 56,76,64,45,67,98,65,54,76,86

          dc.w 55,60,65,70,75,80,85,90,95,100
          dc.w 56,76,64,45,67,98,65,54,76,86

          dc.w 55,60,65,70,75,80,85,90,95,100
          dc.w 56,76,64,45,67,98,65,54,76,86

pic:      blk.b 12240,0
```

6.4. ANREGUNGEN

Dieses Kapitel soll Ihnen weitere Anregungen zum Thema "Spieleprogrammierung" geben. Auf den folgenden Seiten sind einige Vorschläge zur Verbesserung bzw. zur Erweiterung der Programme aufgeführt, die Sie in Band 1 und Band 2 von "Spiele selber programmieren" finden konnten. Mit den bis jetzt erworbenen Kenntnissen müßten Sie in der Lage sein, die folgenden Änderungen vorzunehmen.

1) Zwei- oder Drei-Spielermodus
Unser kleines Spiel ist nur für einen Spieler, der gegen den Computer antritt, ausgelegt. Sie können nun einen zweiten Spieler mit einem in Port 0 angeschlossenen Joystick programmieren. Dazu müssen Sie lediglich die Routinen für die Joystick-Abfrage von Port 1 kopieren und für Port 0 umändern, indem Sie das Register $dff00a verwenden.
Sogar ein dritter Spieler könnte in das Spielgeschehen eingreifen, wenn Sie seine Bewegungen über die Tastatur einlesen. Ebenfalls könnte man den Spieler selbst wählen lassen, mit welchen Tasten er sein Symbol lenken möchte, indem man ihn am Anfang die Tasten selbst definieren läßt. Hat man nun mehrere Spieler gleichzeitig, so müssen natürlich auch die Kollisionen darauf eingerichtet werden.

2) Sternenhimmel Titelbild
Das Beispiellisting Sternenhimmel steht mit unserem Spiel (siehe Band 1) nicht in Zusammenhang. Man könnte sich diesen netten Effekt aber als Titelvorspann zunutze machen. Nachdem man das Spiel gestartet hat, erscheint nicht gleich das Spielfeld, sondern ein schwarzer Bildschirm aus dessen Tiefe Sterne erscheinen. Ebenso könnte man den Titel in den Hintergrund einblenden. Um das zu erreichen, müssen Sie eine zweite Copperliste anlegen. Die richtige Copperliste, die zum Hauptprogramm gehört, wird erst nach dem Druck auf den Joystick aktiviert, dadurch schaltet der Computer zwischen Sternen und Spiel um. Wenn man die Sterne ebenfalls im Dual-Playfield-Modus programmiert, könnte man im zweiten Play-

field den Titel des Spieles einblenden. Im Dual-Playfield-Modus kann man maximal 3 Bitplanes verwenden, das entspricht 8 Farben.

3) Attached Sprites
Da wir nur vier Sprites für unser Spiel verwenden, ist der Einsatz von Attached Sprites für alle vier Symbole möglich. Nur muß beachtet werden, daß auch die Bits für die ungeraden Sprites bei der Kollision gesetzt werden müssen, damit diese zugelassen werden (Register $dff098). Auch ist es dann nicht mehr möglich, mehrere Spieler mit Sprites zu erschaffen, denn es kann nur maximal vier Attached Sprites geben.

4) Soundeffekte
Im Kapitel Sound-Programmierung wurde die Anwendung und das Abspielen von Samples besprochen. Während dem Programm läuft ein Sample als Hintergrund-Musik. Als kleine Steigerung können Sie mit einem Soundsampler mehrere Klänge und Geräusche sampeln und bei einer Kollision abspielen. Ein ganz netter Effekt ist auch eine gesampelte Stimme, die bei einer Kollision gespielt wird.

4) Blitter-Einsatz
Das in Band 1 enthaltene Spielelisting bedient sich in keinster Weise des Blitters, da dieser erst in Band 2 besprochen wurde. Nun sollten Sie aber schon genügend Wissen haben, um den Blitter auch in unserem kleinen Beispiellisting zu verwenden.

Es wird ein kleines Datenfeld nach jeder Explosion des Spielers kopiert: Nämlich das Datenfeld, in dem die Zahl der Leben steht. Diese Aufgabe wird noch vom Prozessor übernommen. Im Beispielprogramm des Kapitels "Einfaches Kopieren" wird dieser Vorgang mit dem Blitter demonstriert. Ersetzen Sie diese Routinen auch im Spiel!

ANHANG

I. ÜBERSICHT DER HARDWARE-REGISTER

Das wohl wichtigste für Spiele-Programmierer sind die Hardware-Register. In diesem Kapitel finden Sie eine Auflistung aller Hardware-Register des Amiga.

NAME	ADRESSE	FUNKTION
BLTDDAT	$dff000	Blitter Destination Data
DMACONR	$dff002	DMA Control Read
VPOSR	$dff004	Vertical Position Read
VHPOSR	$dff006	Vert. and horiz. Position Read
DSKDATR	$dff008	Disk data read
JOY0DAT	$dff00a	Joy-Port 0 data
JOY1DAT	$dff00c	Joy-Port 1 data
CLXDAT	$dff00e	Collision data
ADKCONR	$dff010	Audio Disk Control Read
POT0DAT	$dff012	Pot0 Dat a
POT1DAT	$dff014	Pot1 Dat a
POTGOR	$dff016	Pot data read
SERDATR	$dff018	Serial data read
DSKBYTR	$dff01a	Disk data byte read
INTENAR	$dff01c	Interrupt enable bits read
INTREQR	$dff01e	Interrupt request bits read
DSKPTH	$dff020	Disk pointer high
DSKPTL	$dff022	Disk pointer low
DSKLEN	$dff024	Disk data length
DSKDAT	$dff026	Disk DMA data write
REFPTR	$dff028	Refresh pointer
VPOSW	$dff02a	Vertical Position write
VHPOSW	$dff02c	Vert. and horiz. Pos. write
COPCON	$dff02e	Coprozessor control register
SERDAT	$dff030	Serial port data

SERPER	$dff032	Serial port period
POTGO	$dff034	Pot port data write and go
JOYTEST	$dff036	Joy port Test
STREQU	$dff038	Strobe for hsync with VB and EQU
STRVBL	$dff03a	Strobe for horiz. sync with VB
STRHOR	$dff03c	Strobe for horiz. sync
STRLONG	$dff03e	Strobe for identific. of long horiz. line
BLTCON0	$dff040	Blitter control register 0
BLTCON1	$dff042	Blitter control register 1
BLTAFWM	$dff044	Blitter first word mask for A
BLTALWM	$dff046	Blitter last word mask for A
BLTCPTH	$dff048	Blitter pointer high to C
BLTCPTL	$dff04a	Blitter pointer low to C
BLTBPTH	$dff04c	Blitter pointer high to B
BLTBPTL	$dff04e	Blitter pointer low to B
BLTAPTH	$dff050	Blitter pointer high to A
BLTAPTL	$dff052	Blitter pointer low to A
BLTDPTH	$dff054	Blitter pointer high to D
BLTDPTL	$dff056	Blitter pointer low to D
BLTSIZE	$dff058	Blitter size and start
	$dff05a	nicht belegt
	$dff05c	nicht belegt
	$dff05e	nicht belegt
BLTCMOD	$dff060	Blitter modulo C
BLTBMOD	$dff062	Blitter modulo B
BLTAMOD	$dff064	Blitter modulo A
BLTDMOD	$dff066	Blitter modulo D
	$dff068	nicht belegt
	$dff06a	nicht belegt
	$dff06c	nicht belegt
	$dff06e	nicht belegt
BLTCDAT	$dff070	Blitter C data register
BLTBDAT	$dff072	Blitter B data register
BLTADAT	$dff074	Blitter A data register
	$dff076	nicht belegt
	$dff078	nicht belegt

	$dff07a	nicht belegt
	$dff07c	nicht belegt
DSKSYNC	$dff07e	Disk synchronisation pattern
COP1LCH	$dff080	Copper first location high
COP1LCL	$dff082	Copper first location low
COP2LCH	$dff084	Copper second location high
COP2LCL	$dff086	Copper second location low
COPJMP1	$dff088	Restart Copper first location
COPJMP2	$dff08a	Restart Copper sec. location
COPINS	$dff08c	Copper instruction fetch
DIWSTRT	$dff08e	Display window start
DIWSTOP	$dff090	Display window stop
DDFSTRT	$dff092	Display data fetch start
DDFSTOP	$dff094	Display data fetch stop
DMACON	$dff096	DMA control register
CLXCON	$dff098	Collision control
INTENA	$dff09a	Interrupt enable bits
INTREQ	$dff09c	Interrupt request bits
ADKCON	$dff09e	Audio and disk control
AUD0LCH	$dff0a0	Audio0 location high
AUD0LCL	$dff0a2	Audio0 location low
AUD0LEN	$dff0a4	Audio0 data length
AUD0PER	$dff0a6	Audio0 period
AUD0VOL	$dff0a8	Audio0 volume
AUD0DAT	$dff0aa	Audio0 data register
	$dff0ac	nicht belegt
	$dff0ae	nicht belegt
AUD1LCH	$dff0b0	Audio1 location high
AUD1LCL	$dff0b2	Audio1 location low
AUD1LEN	$dff0b4	Audio1 data length
AUD1PER	$dff0b6	Audio1 period
AUD1VOL	$dff0b8	Audio1 volume
AUD1DAT	$dff0ba	Audio1 data register
	$dff0bc	nicht belegt
	$dff0be	nicht belegt
AUD2LCH	$dff0c0	Audio2 location high

AUD2LCL	$dff0c2	Audio2 location low
AUD2LEN	$dff0c4	Audio2 data length
AUD2PER	$dff0c6	Audio2 period
AUD2VOL	$dff0c8	Audio2 volume
AUD2DAT	$dff0ca	Audio2 data register
	$dff0cc	nicht belegt
	$dff0ce	nicht belegt
AUD3LCH	$dff0d0	Audio3 location high
AUD3LCL	$dff0d2	Audio3 location low
AUD3LEN	$dff0d4	Audio3 data length
AUD3PER	$dff0d6	Audio3 period
AUD3VOL	$dff0d8	Audio3 volume
AUD3DAT	$dff0da	Audio3 data register
	$dff0dc	nicht belegt
	$dff0de	nicht belegt
BPL1PTH	$dff0e0	Bitplane1 pointer high
BPL1PTL	$dff0e2	Bitplane1 pointer low
BPL2PTH	$dff0e4	Bitplane2 pointer high
BPL2PTL	$dff0e6	Bitplane2 pointer low
BPL3PTH	$dff0e8	Bitplane3 pointer high
BPL3PTL	$dff0ea	Bitplane3 pointer low
BPL4PTH	$dff0ec	Bitplane4 pointer high
BPL4PTL	$dff0ee	Bitplane4 pointer low
BPL5PTH	$dff0f0	Bitplane5 pointer high
BPL5PTL	$dff0f2	Bitplane5 pointer low
BPL6PTH	$dff0f4	Bitplane6 pointer high
BPL6PTL	$dff0f6	Bitplane6 pointer low
	$dff0f8	nicht belegt
	$dff0fa	nicht belegt
	$dff0fc	nicht belegt
	$dff0fe	nicht belegt
BPLCON0	$dff100	Bitplane control register 0
BPLCON1	$dff102	Bitplane control register 1
BPLCON2	$dff104	Bitplane control register 2
	$dff106	nicht belegt
BPL1MOD	$dff108	Bitplane Modulo 1

BPL2MOD	$dff10a	Bitplane Modulo 2
	$dff10c	nicht belegt
	$dff10e	nicht belegt
BPL1DAT	$dff110	Bitplane1 data
BPL2DAT	$dff112	Bitplane2 data
BPL3DAT	$dff114	Bitplane3 data
BPL4DAT	$dff116	Bitplane4 data
BPL5DAT	$dff118	Bitplane5 data
BPL6DAT	$dff11a	Bitplane6 data
	$dff11c	nicht belegt
	$dff11e	nicht belegt
SPR0PTH	$dff120	Sprite0 pointer high
SPR0PTL	$dff122	Sprite0 pointer low
SPR1PTH	$dff124	Sprite1 pointer high
SPR1PTL	$dff126	Sprite1 pointer low
SPR2PTH	$dff128	Sprite2 pointer high
SPR2PTL	$dff12a	Sprite2 pointer low
SPR3PTH	$dff12c	Sprite3 pointer high
SPR3PTL	$dff12e	Sprite3 pointer low
SPR4PTH	$dff130	Sprite4 pointer high
SPR4PTL	$dff132	Sprite4 pointer low
SPR5PTH	$dff134	Sprite5 pointer high
SPR5PTL	$dff136	Sprite5 pointer low
SPR6PTH	$dff138	Sprite6 pointer high
SPR6PTL	$dff13a	Sprite6 pointer low
SPR7PTH	$dff13c	Sprite7 pointer high
SPR7PTL	$dff13e	Sprite7 pointer low
SPR0POS	$dff140	Sprite0 start position
SPR0CTL	$dff142	Sprite0 control data
SPR0DATA	$dff144	Sprite0 image data A
SPR0DATB	$dff146	Sprite0 image data B
SPR1POS	$dff148	Sprite1 start position
SPR1CTL	$dff14a	Sprite1 control data
SPR1DATA	$dff14c	Sprite1 image data A
SPR1DATB	$dff14e	Sprite1 image data B
SPR2POS	$dff150	Sprite2 start position

SPR2CTL	$dff152	Sprite2 control data
SPR2DATA	$dff154	Sprite2 image data A
SPR2DATB	$dff156	Sprite2 image data B
SPR3POS	$dff158	Sprite3 start position
SPR3CTL	$dff15a	Sprite3 control data
SPR3DATA	$dff15c	Sprite3 image data A
SPR3DATB	$dff15e	Sprite3 image data B
SPR4POS	$dff160	Sprite4 start position
SPR4CTL	$dff162	Sprite4 control data
SPR4DATA	$dff164	Sprite4 image data A
SPR4DATB	$dff166	Sprite4 image data B
SPR5POS	$dff168	Sprite5 start position
SPR5CTL	$dff16a	Sprite5 control data
SPR5DATA	$dff16c	Sprite5 image data A
SPR5DATB	$dff16e	Sprite5 image data B
SPR6POS	$dff170	Sprite6 start position
SPR6CTL	$dff172	Sprite6 control data
SPR6DATA	$dff174	Sprite6 image data A
SPR6DATB	$dff176	Sprite6 image data B
SPR7POS	$dff178	Sprite7 start position
SPR7CTL	$dff17a	Sprite7 control data
SPR7DATA	$dff17c	Sprite7 image data A
SPR7DATB	$dff17e	Sprite7 image data B
COLOR00	$dff180	Color register 00
COLOR01	$dff182	Color register 01
COLOR02	$dff184	Color register 02
COLOR03	$dff186	Color register 03
COLOR04	$dff188	Color register 04
COLOR05	$dff18a	Color register 05
COLOR06	$dff18c	Color register 06
COLOR07	$dff18e	Color register 07
COLOR08	$dff190	Color register 08
COLOR09	$dff192	Color register 09
COLOR10	$dff194	Color register 10
COLOR11	$dff196	Color register 11
COLOR12	$dff198	Color register 12

FEHLERMELDUNGEN:

Datei konnte nicht geöffnet werden!
Der angegebene Dateiname ist entweder falsch, das Verzeichnis wurde nicht korrekt angegeben, oder das File existiert nicht.

Breite zu groß oder zu klein (1-20).
Es wurde eine unkorrekte Breite angegeben.

Höhe zu groß oder zu klein (1-256).
Die Angabe der Höhe ist falsch.

Planeanzahl zu groß oder zu klein (1-5).
Die Planeanzahl stimmt nicht.

Datei laden	"l"	Es wird ein bereits ins RAW-Format konvertiertes Bild (kein IFF-Bild!) geladen. Dabei sind folgende Angaben zu machen: Name der Datei: Pfad/Name des Bildes Breite in Words: Breite des Bildes in Words (320 Pixel = 20 Words). Höhe: Höhe des Bildes in Zeilen. Anzahl der Planes: Zahl der verwendeten Bitplanes. Neben- (n) oder Unter- (Return) oder x: Hier kann selektiert werden, ob die Bitplanes, die geladen werden, untereinander, oder nebeneinander liegen. Will man ein gerade erst umgewandeltes RAW-Bild laden, liegen die Bitplanes untereinander.
Datei sichern	"s"	Eine fertig konvertierte Datei wird gespeichert. Dabei sind folgende Angaben zu machen: Name der Datei: Pfad/Name des Bildes. Neben- (n) oder Unter- (Return) oder x: Hier wird selektiert, wie das vorliegende Bild abgespeichert werden soll, entweder untereinander oder nebeneinander.
Hilfe	"h"	Gibt eine kurze Information aus.
Bild ansehen	"a"	Mit dieser Funktion kann das zuletzt geladene Bild betrachtet werden.
Ende	"x"	Programmende

1. Zeile der ersten Plane
1. Zeile der zweiten Plane
1. Zeile der dritten Plane
2. Zeile der ersten Plane
2. Zeile der zweiten Plane
2. Zeile der dritten Plane
3. Zeile der ersten Plane
.
.

Einfach gesagt, muß man lediglich die Zeilen umkopieren, sodaß aus dem ersten Schema das zweite Schema wird; diese Arbeit übernimmt der Bitplane-Mixer.

BEDIENUNG:

Das Programm bietet Ihnen einige Funktionen, die ein schnelles Bearbeiten der Bitplanes ermöglichen. Die einzelnen Kommandos werden über die Tastatur eingegeben. Wichtig ist, daß diese Kommandos in Kleinbuchstaben eingegeben werden.

Funktion	Kommando	Beschreibung
dir	"dir"	Gibt den Inhalt des aktuellen Verzeichnisses auf dem Bildschirm aus. Wichtig ist, daß im Root-Verzeichnis der CLI-Befehl "dir" vorhanden ist, da dieser nachgeladen wird. Wird der dir-Befehl nicht gefunden, kann das Directory auch nicht angezeigt werden.

III. BITPLANE-MIXER

EINLEITUNG:

Wenn man ein IFF-Bild ins RAW-Format umwandelt (z.B.: mit dem IFF-CONVERT-UTILITY), werden die Bitplanes untereinander im Speicher abgelegt.

1. Zeile der ersten Plane
2. Zeile der ersten Plane
.
.
.
x. Zeile der ersten Plane
1. Zeile der zweiten Plane
2. Zeile der zweiten Plane
.
.
.
x. Zeile der zweiten Plane
.
.
.

Will man aber mehrere Bitplanes mit nur einem Blitterstart kopieren, müssen diese in einem anderen Format abgelegt sein (siehe dazu Kapitel Tips & Tricks zum Blitter). Die Bitplanes müssen dazu zeilenweise hintereinander abgelegt sein.

Das Bild kann nicht angezeigt werden! Illegale Breite!
Das Bild ist zu breit und wird deshalb vom Programm nicht auf dem Monitor angezeigt. Es wird aber trotzdem richtig ins RAW-Format umgewandelt!

- BILD ANSEHEN

Mit diesem Gadget können Sie ein geladenes Bild betrachten. Falls Sie ein Bild laden, das breiter ist als 320 Pixel, wird es nicht dargestellt, aber natürlich korrekt ins RAW-Format übertragen.

- COLORMAP

Dieser Schalter ist zum Ein- und Ausschalten der Farbtabelle gedacht, welche jedem IFF-Bild beigefügt ist. In dieser Tabelle stehen die Farbwerte des jeweiligen Bildes. Das IFF-CONVERT-UTILITY bietet die Möglichkeit, diese Farbtabelle am Ende des Bildes anzuhängen und mitzuspeichern (Colormap ein), oder gänzlich wegzulassen (Colormap aus). Denn vor allem bei der Programmierung ist oftmals die Farbtabelle nicht notwendig. In der Grundeinstellung ist sie eingeschaltet.

FEHLERMELDUNGEN:

Im Kästchen STATUS werden vom Programm Fehlermeldungen, beziehungsweise andere Programmschritte, angeführt.

Leider nicht genug Speicher vorhanden!
Der Speicherplatz hat nicht ausgereicht, um das angegebene Bild zu laden. Dies kann passieren, wenn man übergroße Bilder laden will oder nebenbei andere Programme laufen hat.

Kein IFF-Format!
Das zu ladende File ist kein IFF-Bild.

Das angegebene File existiert nicht!
Das File wurde nicht gefunden, oder es wurde kein Name angegeben.

im Speicher liegt. Daher verwendet man gleich das RAW-Format. Außerdem werden eigene Programme meistens mit einem Cruncher gepackt, sodaß dieser Platzverlust wieder wettgemacht wird. Für die in diesem Buch beschriebenen Beispiele ist es wichtig, selbsterstellte Bilder vom IFF-, ins RAW-Format umzuwandeln. Ebenso können Sie eigene Schriften und Bilder für Demos erzeugen.
Achtung: Das IFF-Convert-Utility ist **nicht** für die Konvertierung von Bildern mit 256-Farben ausgelegt!

DIE BEDIENUNG

Folgende Funktionen stehen zur Verfügung:

- LADEN

Wenn Sie dieses Gadget anklicken, wird ein IFF-Bild nachgeladen. Der Name muß vorher natürlich in der Zeile NAME DES IFF-BILDES: eingegeben werden. Nach dem Laden ist das Bild schon fertig konvertiert und kann sofort gespeichert werden.

- SPEICHERN

Hier wird das bereits entpackte Bild als RAW-FORMAT abgespeichert. Und zwar unter dem Namen, den Sie in der Zeile NAME DES RAW-BILDES: angegeben haben.

- CONVERT

Diese Funktion vereint die beiden oberen. Das kann oft sehr nützlich sein, wenn man mehrere Laufwerke besitzt, oder das IFF- und das RAW-Bild auf der gleichen Diskette bearbeiten will. Geben Sie zuerst in den dafür vorgesehenen Zeilen die Namen der Bilder ein und drücken Sie CONVERT. Dann wird zuerst das IFF-Bild geladen, umgewandelt und anschließend sofort gespeichert.

II. IFF-CONVERT-UTILITY

EINLEITUNG

Mit dem IFF-CONVERT-UTILITY ist es möglich, IFF-Bilder, die mit einem Malprogramm, z.B. DPaint erstellt wurden, in das RAW-Format umzuwandeln. Nun, welchen Sinn hat das? Ganz einfach: Der Videochip kann nur das RAW-Format verarbeiten. Das bedeutet, daß das Bild im Speicher unseres Computers als RAW-Format vorliegen muß, damit es dargestellt werden kann. Wird ein mit DPaint gezeichnetes Bild abgespeichert, so kommt das IFF-Format zur Anwendung.
Vielleicht ist es Ihnen schon aufgefallen, daß Ihre Bilder trotz Verwendung derselben Auflösung (z.B. LoRes 320x200) unterschiedlich viel Platz auf der Diskette einnehmen. Das liegt daran, daß das IFF-Format die Bilder komprimiert. Dies basiert auf dem Schema der Wiederholung. Befindet sich im Bild eine Zeichenfolge, die gleichlautend ist, wird diese nicht vollständig übernommen, sondern es wird die Anzahl der Wiederholungen, sowie der zu wiederholende Wert angegeben. Ein Beispiel verschafft Klarheit:

Die Zeichenkette z.B.: "$30303030 $30303030" hat eine Länge von 8 Byte. Das IFF-Format übernimmt nicht diese 8 Byte, sondern schreibt ein Byte mit der Anzahl der sich wiederholenden Bytes und danach den Wert der Bytes, in unserem Fall wäre das $30. Damit das erste Byte nicht verwechselt wird, wird es zusätzlich negiert.

Das ist das gesamte Geheimnis des IFF-Formates. Wenn man ein Bild mit vielen einfarbigen Flächen gemalt hat, wird es weniger Platz auf der Diskette verbrauchen, als eines, das viele verschiedene Muster besitzt.

Wenn man programmiert, ist das IFF-Format eher hinderlich, weil man in die eigenen Programme jedesmal ein eigenes Konvertier-Programm einbauen müßte. Außerdem verbraucht das Bild dann mehr Speicherplatz, weil es einmal gepackt und einmal ungepackt

COLOR13	$dff19a	Color register 13
COLOR14	$dff19c	Color register 14
COLOR15	$dff19e	Color register 15
COLOR16	$dff1a0	Color register 16
COLOR17	$dff1a2	Color register 17
COLOR18	$dff1a4	Color register 18
COLOR19	$dff1a6	Color register 19
COLOR20	$dff1a8	Color register 20
COLOR21	$dff1aa	Color register 21
COLOR22	$dff1ac	Color register 22
COLOR23	$dff1ae	Color register 23
COLOR24	$dff1b0	Color register 24
COLOR25	$dff1b2	Color register 25
COLOR26	$dff1b4	Color register 26
COLOR27	$dff1b6	Color register 27
COLOR28	$dff1b8	Color register 28
COLOR29	$dff1ba	Color register 29
COLOR30	$dff1bc	Color register 30
COLOR31	$dff1be	Color register 31

IV DIE PROGRAMMDISKETTE

Um Ihnen das Abtippen der oft sehr langen Programmlistings zu ersparen, haben wir dem Buch eine Diskette beigelegt, auf der nicht nur alle im Buch besprochenen Listings, sondern auch alle dazugehörigen Grafiken, Musikstücke und Tools enthalten sind. Wenn Sie im CLI

dir opt a

eingeben, sollten sich folgende Files und Verzeichnisse auf der Beispieldiskette befinden:

Source (dir)

50 Bobs.s	5Planes1Blit.s
5PlanesMaske1Blit.s	Animationen.s
BobsZoomen.s	CopperAGA.s
EinfachesKopieren.s	EinObjekt.s
FlaechenFuellen.s	Joystick.s
JumpLaufschrift.s	KoordinatenLaufschrift.s
KopierenMitMaske.s	LinienZiehen.s
MehrereObjekte.s	music.s
MusterFuellen.s	ObjekteDrehen.s
SinusLaufschrift.s	SpeicherLoeschen.s
Sternenhimmel.s	

c (dir)

cd	dir

devs (dir)

system-configuration

RAW (dir)

16x16_1pl-breit.raw	16x16_1pl.raw
32x32_5pl.con	Bob32x32_1-breit.raw
Bob32x32_1pl.raw	Bob32x32_5.con

Bob32x32_5.msk
Bob32x32_5.msk.con
Bob32x32_5.raw
Bob32x32_5pl-breit.con
BobZoomMask.con
Bob_16_3pl.con
ColorMask.con
Hintergrund.con
Hintergrund.raw
Muster.raw

IFF (dir)
16x16_1pl-breit.iff
16x16_1pl.iff
32x32_5pl.iff
Bob32x32_1pl.iff
Bob32x32_5pl.iff
BobZoomMask.iff
Bob_16_3pl.iff
ColorMask.iff
Hintergrund.iff
Muster.iff

Demos (dir)
50Bobs
JumpLaufschrift
KoordinatenLaufschrift
ObjekteDrehen
SinusLaufschrift

BITPLANE-MIXER
Boz_3pl.tab
Boz_3pl.tab2
IFF-Convert-Utility
Kords.tab
KreisDat.tab
music SinusDat.tab

V. LITERATURVERZEICHNIS

Falls Sie grundlegende Fragen über den Amiga, seine Bausteine oder die Assemblerprogrammierung haben, können Sie in der unten angeführten Literatur nachschlagen:

1. Amiga Spiele selber programmieren, Band 1
   1996 Verlag Lechner München
   ISBN

2. Amiga Hardware Reference Manual
   1986 Addison Wesley, Inc.
   ISBN 0-201-11077-6

   update version 1989
   ISBN 0-201-18157-6

Grundlagen über die Assemblerprogrammierung und das Amiga-Betriebssystem können sie in den unten angeführten Büchern nachlesen:

3. ASSEMBLERPRAXIS, Niki Laber
   1992 Verlag Gabriele Lechner
   ISBN 3-926858-38-9

4. AMIGA DELUXE PAINT IV, Walter Friedhuber u. Anton Koller
   1992 Verlag Gabriele Lechner
   ISBN 3-926858-33-8

5. The Amiga DOS Manual, (englische Originalfassung)
   1986 Bantam Books
   ISBN 0-553-34294-0

6. AmigaDOS Handbuch
   Deutsche Übersetztung vom engl. Original, M&T ISBN 3-89090-465-3

7. M68000 Familie Teil 1, Hilf/Nausch
1984 tewi
ISBN 3-921-803-16-0

8. M68000 Familie Teil 2, Hilf/Nausch
1984 tewi
ISBN 3-921-803-30-6

9. Die 68000er Serie Buch 1, Nachmann
1986 Elektor Verlag
ISBN 3-921608-42-2

10. Die 68000er Serie Buch 2, Nachmann
1986 Elektor Verlag

STICHWORTVERZEICHNIS

AMIGA SPIELE SELBER PROGRAMMIEREN BAND 2

Mit der Buchreihe „Amiga Spiele selber programmieren" werden sowohl Fortgeschrittene als auch Einsteiger in die Spieleprogrammierung eingeführt. Band 2 beschäftigt sich mit den Programmier-Tricks und -Feinheiten. Den Beginn bildet eine Einführung in die Musikprogrammierung, damit Sie eigene Module in Ihr Spiel integrieren können. Ein großes Kapitel ist dem Blitter, einem der wichtigsten Bausteine des Amigas, gewidmet. Anhand von vielen praktischen Programmbeispielen wird das Zusammenspiel der einzelnen Chips, wie Blitter und Copper erklärt.

Programmbeispiele: das AGA-Chipset und die Darstellung von 256 Farben, Sinuslaufschrift, Laufschrift nach Koordinatentabelle, Jump-Laufschrift, mehrere Objekte gleichzeitig bewegen, Bobs zoomen, Sternenhimmel uvm.

Alle Programmlistings, Grafiken und Musikstücke, sowie einige nützliche Tools finden Sie auf der beiliegenden Programmdiskette.

Verlag Gabriele Lechner
Bodenseestraße 91
81243 München

ISBN 3-926858-64-8
DM 39,- SFr 36,- öS 285,-
inklusive 1 Diskette